8-9-2020

Dynamic Network Configurations for Functionality and Security

Devon Callahan
University of Connecticut - Storrs, devon.callahan@uconn.edu

Dynamic Network Configurations for Functionality and Security

Devon M. Callahan, Ph.D.

University of Connecticut, 2020

ABSTRACT

Networks are designed with functionality, security, performance, and cost in mind. Flows should be served while controlling risk due to attackers. Configuration is time intensive and largely static until a major new vulnerability or service requirement forces change. We address this problem with an autonomous framework consisting of Observe, Orient, Decide and Act phases and look to optimization techniques for solutions to the Orient and Decide phases.

Our first solution explores opportunities to improve network Quality of Service by combining a single flow routing solutions with a global multi-flow solution in a hybrid manner. In order to evaluate the quality of our solutions we implement an autonomous framework which generates the routing solution in a software defined network.

We then explore two additional solutions that address both functional and security requirements and explore the trade-off of modeling and implementation choices for this problem. These two solutions innovate in modeling security risk in a way that is amenable to optimization and in the evaluation of the quality of the resulting configurations.

Devon M. Callahan,
University of Connecticut, 2020

Our framework allows an enterprise to automatically reconfigure their network upon a change in functionality (shift in user demand) or security (publication or patching of a vulnerability).

The primary contributions of this work are two-fold: 1) the formulation and integrations of methods to address network Quality of Service and security in an autonomous framework and 2) detailed evaluation of these methods combining both emulation and simulation.

Dynamic Network Configurations for Functionality and Security

Devon M. Callahan

M.S., University of South Carolina, 2009

B.S., Methodist University, 2005

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

at the

University of Connecticut

2020

APPROVAL PAGE

Doctor of Philosophy Dissertation

Dynamic Network Configurations for Functionality and Security

Presented by

Devon M. Callahan, B.S., M.S.

Major Advisor _____
Benjamin Fuller

Associate Advisor _____
Laurent Michel

Associate Advisor _____
Amir Herzberg

University of Connecticut

2020

ACKNOWLEDGMENTS

I am forever indebted to my family, for taking this journey with me. I certainly would not be where I am with out your love and support. Being a family to a Soldier you have made sacrifices that most would not understand and we are stronger because of it. Your support and understanding over the past three years has enabled me to achieve my academic goals. Specifically to my wife, Renee, you are an amazing woman and I am so happy to have you as a partner in life.

Thank you to my major advisor, Dr. Benjamin Fuller for providing a great environment to academically grow. You have trusted me to pursue my academic interests with your full support. I consider myself extremely lucky to have a mentor whom I trust without reservation, knowing my academic development is always top priority.

Thank you Dr. Laurent Michel for supporting my academic development and for being a great person to work with. I am amazed that one person can accomplish so much and be dedicated to the success of so many.

Thank you Dr. Amir Herzberg for your commitment to excellence. You have provided extremely valuable guidance and feedback in support of my academic development. Thank you for the time you have invested in me.

My sincerest appreciation to my colleagues (friends) who have spent countless hours making this work possible. I am fortunate to have worked with amazingly smart and kind people who have been more patient with me than I deserve. I have no doubt that with out their support I would not have succeeded in this endeavor.

Contents

Chapter 1

Introduction

Network administrators must balance functionality, performance, security, cost and other industry specific requirements. Implementation of this vision becomes complex when considering that priorities are often antagonistic. An an improvement of one property may negatively impact another. Striking the appropriate balance between opposing requirements is complex. In order for our networks to be responsive to user demands and resilient to threats they must be able to strike this balance quickly.

Tools assist administrators with this complex task: existing work assesses network reachability [65], wireless conflicts [90], network security risk [106, 126], and load balancing [114, 122]. These tools assess the quality of a potential configuration. Unfortunately, current tools suffer from three limitations:

1. Tools assess whether a single property is satisfied, making no recommendation if the property is not satisfied. This leaves IT personnel with the task of deciding how to change the network.
2. Tools assess networks with respect to an individual goal at a time. This means

a change to satisfy a single property may break another property. There is no guidance for personnel on how to design a network that meets the complex and often conflicting network requirements.

3. Tools do not react to changing external information such as the publication of a new security vulnerability.

A more desirable scenario is a highly integrated and flexible system, providing solutions that meet all possible requirements with the ability to rapidly apply those solutions. We address this problem with a autonomous solution consisting of Observe, Orient, Decide and Act phases and look to optimization techniques for solutions to the Orient and Decide phases.

Our first solution explores opportunities to improve network Quality of Service by combining current single flow routing solutions with a global multi-flow solution in a hybrid manner. In order to evaluate the quality of our solutions we implement a autonomous framework that generates benchmarks for input to our Hybrid solver. The framework generates an SDN network from the description in the benchmark, translates the high-level model solutions into routable Openflow tables, and evaluate the resulting configuration with IPv4 traffic.

We then explore two additional solutions that address both functional and security requirements and explore the trade-off of modeling and implementation choices for this problem. These two solutions innovate in modeling security risk in a way that is amenable to optimization and in the evaluation of the quality of the resulting configurations.

The network configuration problem is difficult, networks are large and must be updated quickly. Our approach has the most promise for software-defined networks

which can easily change their logical configuration. Our framework allows an enterprise to automatically reconfigure their network upon a change in functionality (shift in user demand) or security (publication or patching of a vulnerability). The primary contributions of this work are two-fold: 1) the formulation and integrations of methods to address network Quality of Service and security in a autonomous framework and 2) detailed evaluation of these methods combining both emulation and simulation.

1.1 Background

1.1.1 Autonomous Networking

Observe, Orient, Decide, Act (OODA) is a decision making framework created by COL William Boyd of USAF in the 1987 to explain combat effectiveness of air to air combat during the Korean War [21]. We use this framework and apply the principals of autonomic computing: self configuration, healing, optimization and protection [63] to our network problem. Other frameworks exist. MAPE-K (Monitor, Analyze, Plan, Execute and Knowledge) is a reference architecture in use by IBM in the field of autonomic computing [31]. We choose to use OODA framework for our discussion throughout based on its wide spread application in command and control systems beyond autonomous networking.

1.1.2 Data Center Networks

Data Center Networks (DCN) host, process and analyze data in financial, entertainment, medical, government and military sectors. The services provided by DCNs must

be reliable, accurate and timely. Services provided by DCNs (and the corresponding traffic) are heterogeneous [84]. The network must adapt to changing priorities and requirements while protecting from emerging threats. They scale to thousands of servers linked through various interconnects. Protocols used for these services are split roughly 60 percent web (HTTP/HTPS) and 40 percent file storage (SMB/AFS) [17]. The interdependence of device configurations make modifying any single configuration difficult and possibly dangerous for network health. A seemingly simple update can cause significant collateral damage and unintended consequences [69].

1.1.3 Software Defined Networks

The network fabric is changing with the advent of Software Defined Networking (SDN) [70]. SDNs separate the control logic from the data forwarding which is integrated in traditional networks. SDN provides a control plane which is administered by a network controller and a separate forwarding plane which resides on the switches and routers. SDNs are flexible and programmable networks that can adapt to emergent functional or performance requirements. Openflow [83] is a common open source software stack used to communicate between SDN controllers and SDN enabled devices (switches and routers). Unfortunately, Openflow is 1) a relatively low level language requiring multiple rules to implement simple concepts 2) difficult to ensure correct application of policies from different modules (routing, access control) [44] Researchers have proposed high-level languages and compilers [15, 44, 66, 101] that bridge the semantic gap between network administrators and the configuration languages used by SDN devices. These languages focus on *compositional* and *parametric* SDN software modules that execute specific micro-functions (e.g., packet forwarding,

dropping, routing, etc.). The use of a high level language is prompted by a desire to be able to *select*, *instantiate* and *compose* SDN modules with guarantees.

1.2 Overview of Technical Works

1.2.1 Chapter 3 GOSH

In Chapter 3 we introduce a framework GOSH - Global Optimization in Software-defined networks a Hybrid approach. This framework is designed to improve network QoS by combining techniques from DCN and backbone networking fields. The majority of QoS routing solutions in DCN networks are solved in an online manner solving single flow requests [18]. This approach has the benefit of being fast (msec) but with the possible loss of accuracy. Conversely in internet backbone routing there are offline approaches that solve for multiple flows that are slower (seconds-minutes) but provide global optimality [26, 119, 125].

We combine the above *single flow* and *multi flow* strategies, using single flow solutions to route emerging demand and periodically implementing multi-flow solution to ensure that the quality of service in the network does not degrade significantly over time. We develop an autonomous framework using Mininet, an SDN emulation platform, to deploy and to evaluate the feasibility of this approach and the quality of our solutions.

1.2.2 Chapter 4 DocSDN

This work introduces a new optimization framework that finds network configurations that satisfy multiple (conflicting) requirements. Our framework is called DocSDN (Dynamic and Optimal Configuration of Software-Defined Networks). DocSDN searches for network configurations that simultaneously satisfy functionality and security requirements. DocSDN is organized into layers that consider different properties. The core of DocSDN is a multistage optimization that decouples search on "orthogonal" concerns. The majority of the technical work is to effectively separate concerns so the optimization problems remain tractable.

1.2.3 Chapter 5 Fashion

In Chapter 5 we develop an optimization framework we call Fashion (Functional and Attack graph Secured HybrId Optimization of virtualized Networks). Similar to the work in the previous chapter Fashion considers both functionality and security in developing network configurations. The *functional* layer treats network traffic as a multi-commodity data flow problem and provides the logic to route flows. To evaluate risk we use Attack Graphs and develop an *approximation* of a prior measure described by Wang et al. [121] that is amenable to quick evaluation using integer linear programming. The *security* layer then integrates the risk of a configuration to create a joint model between the two layers. This work differs from the work presented in Chapter 4 in that Fashion's risk model, through the use of Attack Graphs accounts for an adversary's ability to pivot in the network. In DocSDN each node assumes a fraction of the risk of any node with which they share a path.

1.3 Outline of the Dissertation

This dissertation is organized as follows:I n Chapter 2 we provide details common to all evaluations, In Chapter 3, we look a improving network QoS by combining techniques from DCN and backbone networking fields. In Chapter 4, we discuss the benefits and the challenges of integrating security and functionality models to solve the network configuration problem. In Chapter 5 we implement a combined module for functionality and security with innovations of security risk modeling.

1.4 Publications

Conference papers that are accepted and published with primary authorship include:

1. [25] **D. Callahan**, P. Shakarian, J. Nielsen, and A. N. Johnson, "Shaping operations to attack robust terror networks," in *2012 International Conference on Social Informatics*. IEEE, 2012, pp. 13–18.

2. [33] T. Curry, **D. Callahan**, B. Fuller, and L. Michel, "Docsdn: Dynamic and optimal configuration of software-defined networks," in *Australasian Conference on Information Security and Privacy*. Springer, 2019, pp. 456–474.

   Currently submitted conference papers with primary authorship include

3. [24] **D. Callahan**, T. Curry, D. Davidson, H. Zitoun, B. Fuller, and L. Michel, "Fashion: Functional and attack graph secured hybrid optimization of virtualized networks," *ESORICS*, 2020.

4. **D. Callahan**, T. Curry, D. Davidson, H. Zitoun, B. Fuller, and L. Michel,

"Gosh - Global Optimization in Software-defined networks a Hybrid approach," *IEEE INFOCOM* , 2021

Conference papers that are accepted and published with co-authorship include:

5. [109] P. Shakarian, P. Roos, **D. Callahan,** and C. Kirk, "Mining for geographically disperse communities in social networks by leveraging distance modularity," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '13. New York, NY, USA: ACM, 2013, pp. 1402–1409. [Online]. Available: `http://doi.acm.org/10.1145/2487575.2488194`

6. [110] P. Shakarian, G. I. Simari, and **D. Callahan**, "Reasoning about complex networks: a logic programming approach," MILITARY ACADEMY WEST POINT NY, Tech. Rep., 2013.

Chapter 2

Evaluation

To the best of our knowledge, no standard benchmarks for evaluation of network Functionality and Security solutions exist. For the purpose of this research, we created a benchmark suite with over 600 instances that models a data center topology, its traffic patterns and utilization rates, along with a realistic representation of dispersed network vulnerabilities. The elements of our evaluation topology and benchmark generation common to all chapters follows. Descriptions found in separate chapters are specific to that chapter.

2.1 Topology

A fundamental component of our work is the separation of the physical and logical networks. Our framework has potential in applications where many different logical topologies are possible from a single physical topology. The true benefits of combining constraint programming with SDN is realized on the logical routing topology.

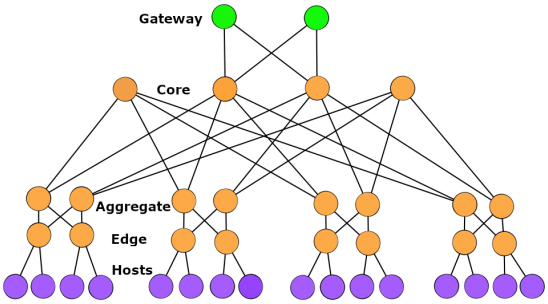


FIGURE 2.1: Order 4 Fat-tree with 2 gateway switches at the top and 2 hosts per edge switch.

The physical network is still very much relevant to solving this problem as it provides us with the set of possibilities. We assume that the physical network is static, composed of possibly miles of cabling connecting layers of networking devices across networks contained in a single room to spanning the globe. The design considerations of these physical networks is important and can greatly improve network performance if properly engineered but is not the focus of our work.

Physical topology is an input to our framework and the empirical evaluation is based on a popular topology: Fat-Tree [5]. An example instance of Fat-Tree we use is shown in Figure 2.1. The network design avoids bottlenecks through multiple equal capacity links between layers thus not necessitating expensive aggregation and core switches. This design uses four layers of switches: gateway, core, aggregate and edge. The edge switches serve as top-of-rack switches and are where our hosts connect. A k-port switch yields an order k Fat-Tree with k *pods* and k switches per pod. The resulting topology provides multiple equal cost paths between hosts.

2.2 Traffic

The network demand module defaults to the traffic pattern described in the recent Global Cloud Index (GCI) report [28] which provides breakdown of data center traffic seen globally. The benchmarks that we generated contain 70% internal traffic and 30% external traffic. To generate internal traffic, pairs of hosts are selected randomly until the number of desired flows is reached, we do not allow repeated pairs. Our benchmarks all contain bi-directional traffic, so for every pair of hosts selected (i, j) for a new demand pairing, we also generate a demand in the reverse (j, i) order.

We model external demand by creating a single external source. This external source is connected to the gateway device and acts as a source/sink for all external traffic. Then each external demand is generated by selecting a single internal host and pairing it with the external source. The external host is not considered regarding capacity constraints for solving or evaluating the model.

Demand is considered constant and measured in the number of Mbps that is required between the source and destination. Research shows that there exists heavy-tailed distributions for the volume and size of data flows [7]. There are generally small and large sized flows with 90% of the traffic volume being small and 10% large [132], our benchmarks follow this distribution, generating small flows between 1-100 Mbps, and large flows 100-500 Mbps.

The generator has a **Traff** input parameter, which determines the number of traffic types generated in the demand. The generator labels each flow with a random abstract traffic type from the pool of traffic types which can be later converted an application layer protocol (HTTP, HTTPS, SMB) or transport layer protocol (TCP,UDP) depending on the desired evaluation. Concretely, in our evaluation in

Chapter 3 we map to UDP and TCP using the iperf3 traffic generation tool.

2.3 Benchmark generator

Each benchmark has 2-3 main components. Benchmarks used in Chapter 3 and Chapter 4 have a network topology and traffic component. The benchmarks used for evaluation in Chapter 5 add a Vulnerability component which is discussed in detail in the related chapter. The following are input parameters for our benchmark generator relating to the network:

Pods the number of pods in the topology which determines the number of switches generated.

Speed the port speed of each switch.

Mem the amount of TCAM memory available. TCAM is particularly important for our evaluation because it can be a limiting factor on the number of flow rules in each switch routing table.

Direct whether to make links directed or undirected.

Hosts the number of hosts per edge switch in the network.

The following are input parameters for our instance generator relating to the traffic demand:

Flows the number demand flows per host in the benchmark.

BiDir if this flag is set we duplicate demand pairings in reverse order.

Traff the number of traffic types.

Network Topology					Network Traffic				Vulnerabilities			
pod	Devices	Hosts	Switch	Links	#traffic		#flows		Exploits		AG edges	
					min	max	min	max	min	max	min	max
2	10	4	6	9	1	3	8	80	1	10	18	165
4	37	16	21	52	1	3	32	320	1	40	258	2350
6	100	54	46	171	1	3	108	1080	5	108	2928	26463
8	209	128	81	400	1	3	256	2560	12	250	16422	147627
10	376	250	126	775	1	2	500	1500	12	400	62537	250765
12	613	432	181	1332	1	3	866	866	21	105	186682	1677306

TABLE 2.1: Benchmark data, 649 instances. The number of links represents the number of bi-directional links in the network. The #traffic column represents the number of distinct traffic types.

Extrn the percentage of the traffic involving an external host. We use a single external host as a super-sink for all external traffic.

Add the amount of additional traffic to generate. This is used in our dynamic testing.

For example, in the Fat-tree 4 network there are 16 hosts, setting the input parameter [Flows] = 1, and setting the [BiDir] flag we have 32 flows (16 x 1 fph x 2 bi-dir traffic). Please see Chapter 5 , for a discussion on Vulnerability generation.

2.4 Benchmarks

A high level breakdown of the benchmark characteristics can be found in Table 2.1.

Chapter 3

Functionality - A Hybrid Approach

3.1 Introduction

Traffic Engineering (TE) is the practice of measuring, modeling, and controlling traffic [2]. We focus on optimizing routing functions by directing traffic in the most "effective" way. Software Defined Networking (SDN) provides the flexibility to optimize routing functions due to the centralized view of the network state and control of the forwarding policies which were previously decentralized. Thus, leveraging SDN provides opportunities to improve on existing TE practices.

The focus of this work is responding to changes in network demand, which are represented as new flows. Our goal is to route these flows in a way that maximizes the overall quality of service in the network. Optimally utilizing the existing resources reduces how much the network must be "over-provisioned" to deal with unexpected demand. To illustrate how traffic can unexpectedly change, consider the shift/growth of internet traffic during the COVID-19 pandemic. AT&T, one of the largest carriers

of internet traffic, reported a 25% increase in traffic during the work week from March to April [42]. This equates to an additional 71 petabytes per day and is more than their entire traffic load in 2014 (56 petabytes per day).

Most TE techniques to determine route selection for a new flow are incremental. That is, when a flow arrives, a route is selected for the duration of the flow. Most of these techniques can be seen as a variant of a Weighted Constrained Shortest Path (CSP) problem [18]. The variety in techniques can be largely expressed as differences in constraints and the objective. For example, edge weights can be based on physical limitations such as distance and propagation delay [118], or current network states such as congestion [37], current flow satisfaction ratios [77] and packet loss on links [124]. These works can be split into heuristic approaches [6, 47, 71] and optimization approaches that provide formal guarantees [74] [127] [37].

At first glance, it seems that any routing decision mechanism must operate on a single flow at a time and must be very fast (taking at most milliseconds). However, this raises two primary questions:

1. How much waste results in routing flows individually?

2. Does the emergence of SDN provide an opportunity to supplement this approach with a multi-flow routing approach?

In the Internet backbone, these are solutions that optimize multiple flows simultaneously [26, 119, 125]. These solutions are highly tailored to Internet Backbone implementations which have fewer nodes and QoS metrics are based on static properties such as geographic separation.

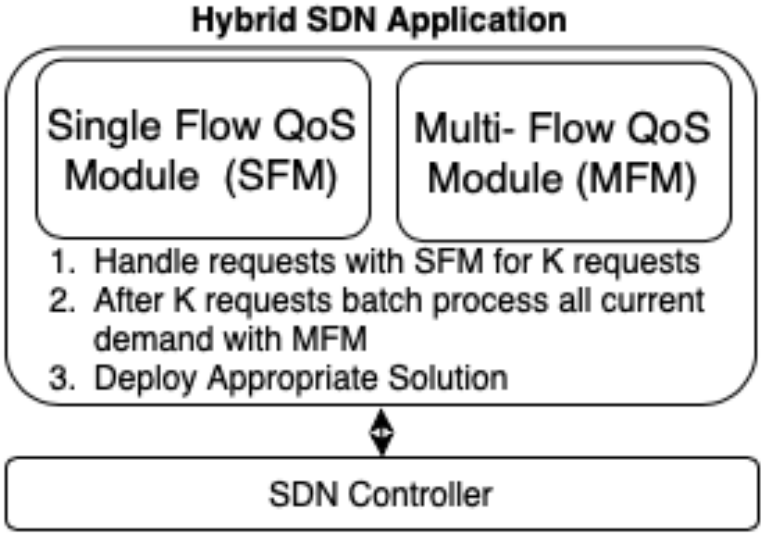


FIGURE 3.1: Integration of a Single Flow Solver and Multi-Flow Solver in a SDN Application

3.1.1 Our Contribution

We combine the above *single flow* and *multi flow* strategies, leveraging the strength of each to offset the inherent weakness of the other. Such an approach is complementary to routing flows as they arrive. Our approach is called GOSH for Global Optimization in Software-defined networks, a Hybrid approach. GOSH consists of the following 3 components:

1. Use *single-flow* route selection as new flows arrive (Figure 3.2a) ,
2. Once a set of k single-flows have arrived recompute a *multi-flow* optimum,
3. If this new *multi-flow* optimum improves on the single-flow solution by "enough," deploy this multi-flow optimum (Figure 3.2b).

This approach, depicted in Figure 3.1, provides the flexibility of quickly solving individual flows as they arrive, while also leveraging current optimization technolo-

gies to ensure that the quality of service in the network does not degrade significantly over time. Moreover, by delaying a global routing update until the point when quality of service can be significantly improved, we avoid an over-engineering of the network routing which would perhaps negatively impact the network's functionality more than a sub-optimal routing configuration would. Traffic patterns may emerge over time identifying certain flows as more long term due to the service they provide such as VoIP or video streaming. We naturally want to wait to perform any global optimization until such a time as these long term flows emerge.

In this work, we show the promise of this approach by:

1. Proposing a mixed integer programming model, denoted Gos (Global Optimization Solver), and combining it in a Hybrid (GOSH) manner with an existing single flow solver, SP (Shortest Path).

2. Evaluating GOSH using TCP traffic in an emulated OpenFlow Network using Mininet [73].

SP acts as our single flow solver and generates the shortest paths between all possible pairs in the graph [108]. In our framework, SP handles all route selection for immediate routing until we generate and implement a global solution with Gos. The novelty of our work is the development of Gos and the development of the hybrid implementation. The single flow solver is modular and can be augmented with other single flow solutions (such as those referenced above).

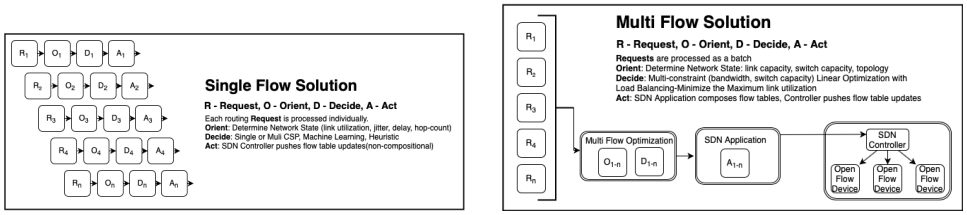
Multi-Flow Solver Gos is a mixed integer programming model which creates a multi-commodity flow quadratic optimization problem based on the prescribed demands in the network. After this, Gos searches for network configurations which not

only solve the flow problem, but also minimize the sum of squares of the link and router utilizations, as well as the sum of the maximum link and router utilizations in the network. This objective effectively provides load balancing within the network. Prior works describe optimization models which optimize only single flows for quality of service or optimize multiple flows and minimize link utilization [3, 74], but to our knowledge there has not been a model such as GOS, which simultaneously optimizes for multiple flows on a full SDN network while minimizing both the maximum link and router utilizations while using destination routing. Prior work assumes that network devices take use both source and destination to make routing decisions. Our model allows only the destination to be used for routing packets.

Once such a solution is obtained, our SDN Application translates the model output into a configuration for an SDN controller (our experiments use Frenetic [43]), then uses the controller to push flow table rules to the SDN devices in the network.

Evaluation We evaluate the performance of our solutions: 1) in a static demand manner to ensure the routing solution delivers the expected throughput, and 2) in a dynamic demand manner paired with a single flow solver SP routing module to show that the QoS can be improved by re-optimizing. Specifically regarding QoS, we consider throughput as a function of the number of bytes delivered per unit of time. We explore the associated costs and benefits of this implementation in Mininet, a software-defined networking emulation framework using Frenetic [43] as the SDN controller.

We evaluated the solution quality of the GOSH framework with networks with up to 370 nodes and over 1500 flows of TCP traffic. For networks of 37 nodes and



(A) Single flow (B) Multi flow

FIGURE 3.2: A The preponderance of existing QoS routing systems: a single flow solution handles each request individually. Figure 3.2b methodology for GOS multiple flows are provided in order to periodically re-establish the globally optimal multi-flow solution

34 flows we solve in half a second and for networks with over a hundred nodes and over 300 flows we solve just over 20 seconds on average. We saw on average 10-40% improvements in throughput between SP and GOS solutions with higher returns as demand approaches network capacity.

Organization The rest of this work is organized as follows. Section 4.2 introduces further related work, Section 3.3 describes our GOS model and the overall GOSH framework, Section 3.4 describes our evaluation setup with results in Section **??**, and finally Section 3.8 concludes.

3.2 Background and Related Work

The primary focus of our work is SDN traffic engineering as it relates to network QoS. As described in the Introduction, the majority of research in this area involves single demand flow solutions for a source - destination pair solved as a CSP. Shortest path problems are often found as a component of the constrained shortest path problems

which are common place in QoS routing for SDN [18]. Here we provide an overview of metrics, single flow solutions, and finally multi-flow solutions.

Metrics There are a range of QoS metrics used in determining the cost of each edge in a shortest path. These metrics can be based on physical limitations, distance and propagation delay [118]. QoS metrics can also describe the current network state, such as the congestion on the network as determined by polling the network switches [37]. Lin et al. [77] measure current flow satisfaction ratios, while packet loss on links is measured in [124]. Zhang et al. [131] insert echo messages and use the latency of these messages to determine edge quality. Similarly, van Adrichem et al. [120] design specific probe packets between measurement points in the network and use timing. These approaches use polling and active measurement techniques. Our solution focuses on the physical limitations of the network and can be augmented with measurement.

Single Flow Solutions Layeghy et al. [74] built a constraint programming model that determined the optimum way to route a new flow, producing solutions in under 400 ms. However, this work did not evaluate the performance of their solutions in a SDN, only the model solve time was considered. Kucminski et al. [71] modified a least cost path algorithm to identify the path that currently has the least load as determined using SDN switch port counters. Egilmez et al. [37] split traffic into two categories: delay sensitive or delay insensitive. Delay insensitive traffic is always routed using shortest path routing. However, when congestion is detected delay sensitive traffic has its path recomputed using a LARAC (Lagrange Relaxation based Aggregated Cost) which solves a Dijkstra's shortest path algorithm with traffic delay constraints. Note their solution is a relaxation and thus heuristic. Note that their technique is not designed to preemptively avoid congestion. Additionally, by only re-routing delay

sensitive traffic it is unclear what impact not considering non-delay sensitive traffic has on over all quality of service.

Multi-Flow Solutions Capelle et al. [26] explore both single flow point-to-point and multicast solutions for network virtualization. Their model includes constraints for link bandwidth and switching capacity as network resources. However they do not consider multiple point-to-point flows in a solution but rather single multicast solutions. In this work the authors allow a 5% gap from optimal as satisfactory. For networks of similar size networks, we are able to find optimal solutions for the multi-flow problem (34 flows) in less than a second while they report solution times of a few seconds for each individual point-to-multipoint flow. Tomonic and Radusinovic [119] use a heuristic approach based on propagation delay and returning the k shortest paths between ingress and egress routers. The authors consider a backbone SDN environment and make the assumption that propagation delay caused by geographic separation will dominate traffic delay. Their work considers traffic as either delay sensitive in which case they attempt to provide QoS or non-delay sensitive in which case routed only if there is sufficient guaranteed bandwidth. Unfortunately, their QoS method of using propagation delay would most likely not translate beyond backbone networks, considering the generally shorter distances between nodes. Additionally, using a shortest path algorithm while not considering current network congestion is likely to result in bottlenecks [123]. Yu et al. [125] use deep reinforcement learning to configure backbone SDN networks. They use the ML model to optimize routing for network performance metrics, such as delay and throughput. To evaluate this framework, the authors built a simulation network environment of 25 nodes using OMNeT++ and compared configurations against routing generated either randomly or using OSPF. Their evaluations focus mainly on delay as the network performance

metric, with some experiments focusing on throughput over different traffic loads. While this approach shows much promise for backbone networks it is unclear how it would perform on networks of larger scale.

3.3 The GOSH framework

This section details the GOSH framework which is described in Figure 3.2. As a reminder, the primary question of this work is:

> Can single-flow problem QoS solutions be augmented with a periodic multi-flow solution?

We view the global solution as "ground truth," something that is accurate for a moment in time before new demands emerge and the network state changes. From this global solution individual demands can be solved with an existing single-flow solution until some time where it is favorable to reacquire the ground truth. Our proposed solution is a way of periodically pushing back to an optimum and ensuring that individual decisions have not been harmful. It is meant to be complimentary to existing solutions that will continue to be deployed to handle emerging demands and paired with our multi-flow solution to re-optimize the global network configuration. We first detail the multi-flow constraint model and then our implementation in an SDN application.

3.3.1 Multi-Flow Solver

This section describes the optimization model used to obtain network configurations that uphold quality of service. GOSH uses a mixed integer programming to model

multi-commodity network flow and correlates improved quality of service with lower global maximum link and router utilizations. In mixed integer programming, the four primary components are Inputs, Variables, Constraints, and an Objective function. These are listed below.

Inputs

$\mathcal{R}$ – the set of SDN devices (routers).

$\mathcal{H}$ – the set of hosts (machines) on the network.

$\mathcal{G} \subset \mathcal{R}$ – the set of external gateway devices in the network.

$\mathcal{D} = \mathcal{R} \cup \mathcal{H}$ – the set of all network devices.

$\mathcal{L} \subset D \times D$ – the set of all network links.

$\mathcal{F}$ – the set of tuples $(h, k) \in \mathcal{D} \times \mathcal{D}$ defining desired traffic flows from source device h to sink device k.

s^* – an artificial node in the network used as the global source of all flows.

t^* – an artificial node in the network used as the global destination of all flows.

$Q(l) : \mathcal{L} \to \mathbb{R}$ – the capacity of link l.

$K(i) : \mathcal{R} \to \mathbb{R}$ – the throughput of network device i.

$\delta^-(i) : \mathcal{D} \to 2^{\mathcal{D}}$ – the set of vertices with outbound arcs leading to vertex i.

$\delta^+(i) : \mathcal{D} \to 2^{\mathcal{D}}$ – the set of vertices with inbound arcs originating at vertex i.

$\mathsf{src}(f) \in \mathcal{D}$ – the network device source of flow f.

$\mathsf{dst}(f) \in \mathcal{D}$ – the network device destination of flow f.

$\mathsf{q}(f) : \mathcal{F} \to \mathbb{R}$ – the quantity of data attributed to flow f.

Variables

$\rho^f_{i,j} \in \{0, 1\}$ – for every $f \in \mathcal{F}$ and every $(i, j) \in \mathcal{L}$, indicates whether link (i, j) carries flow f.

$d^k_{i,j} \in \{0, 1\}$ – for a flow destination k, indicates whether device i sends traffic bound

for k to device j.

$\mu_{i,j} \in \mathbb{R}^+$ – for every $(i,j) \in \mathcal{L}$, the utilization of link (i,j)

$\upsilon_r \in \mathbb{R}^+$ – for every $r \in \mathcal{R}$, the utilization of router r

$\mathcal{M}_l \in \mathbb{R}^+$ – the maximum link utilization in the network.

$\mathcal{M}_r \in \mathbb{R}^+$ – the maximum router utilization in the network.

Constraints

The first part of the model captures the networking side of the model, including how to route flows, respect capacities of devices and how to block specific flows.

$$\rho^f_{s^*,src(f)} = 1 \ , \ \forall f \in \mathcal{F} \tag{3.1}$$

Equation 4.1 forcibly spawns each flow in the network at its network machine source.

$$\sum_{j \in \delta^-(i)} \rho^f_{j,i} = \sum_{k \in \delta^+(i)} \rho^f_{i,k} \ , \ \forall f \in \mathcal{F}, \forall i \in \mathcal{D} \tag{3.2}$$

Equation 4.6 depicts the flow balance equations that dictate that, at every network device, every inbound flow must also be outbound.

$$\rho^f_{i,j} = 0, \ \forall f \in \mathcal{F}, \forall i \in \mathcal{H} \setminus \{\mathsf{src}(f), \mathsf{dst}(f)\}, \forall j \in \delta^+(i) \tag{3.3}$$

$$\rho^f_{\mathsf{dst}(f),j} = 0, \ \forall f \in \mathcal{F}, \forall j \in \delta^+(i) \setminus \{t^*\} \tag{3.4}$$

Equation 3.3 prevents any host from routing traffic for any flows in which it is not involved. Equation 3.4 states that flow destinations can only forward traffic to the

flow sink by preventing the use of any other outbound arc.

$$\sum_{\delta^+(i)} d_{i,j}^k \leq 1 \ , \ \forall i \in \mathcal{N}, \forall k \in \mathcal{H} \tag{3.5}$$

$$d_{i,j}^{\mathsf{dst}(f)} \geq \rho_{i,j}^f \ , \ \forall f \in \mathcal{F}, \forall (i,j) \in \mathcal{L} \tag{3.6}$$

Equations 3.5 and 3.6 enforce destination routing.

$$\sum_{f \in \mathcal{F}} (\rho_{i,j}^f \cdot q(f)) \leq \mathcal{M}_l \cdot Q(i,j), \ \forall (i,j) \in \mathcal{L} \tag{3.7}$$

$$\sum_{f \in \mathcal{F}} \sum_{k \in \delta^-(i)} (\rho_{k,i}^f \cdot q(f)) + \sum_{f \in \mathcal{F}} \sum_{j \in \delta^+(i)} (\rho_{i,j}^f \cdot q(f)) \leq \mathcal{M}_r \cdot K(i), \ \forall i \in \mathcal{R} \tag{3.8}$$

Equation 3.7 simply models the bounds on link capacities, allowing for the capacities to be scaled by the maximum link utilization in the network. Equation 3.8 plays a similar role for the device capacities.

$$\mu_{i,j} = \frac{1}{Q(i,j)} \sum_{f \in \mathcal{F}} \rho_{i,j} \cdot q(f), \ \forall (i,j) \in \mathcal{L} \tag{3.9}$$

$$\upsilon_i = \frac{1}{K(i)} \left(\sum_{f \in \mathcal{F}} \sum_{k \in \delta^-(i)} (\rho_{k,i}^f \cdot q(f)) + \sum_{f \in \mathcal{F}} \sum_{j \in \delta^+(i)} (\rho_{i,j}^f \cdot q(f)) \right), \ \forall i \in \mathcal{R} \tag{3.10}$$

Equations 3.9 and 3.10 calculate, respectively, link and router utilizations.

Objective

The objective function in this model is the sum of squares of the link and router utilizations, along with the maximum link and router utilizations in the network.

FIGURE 3.3: SDN Application. The flow solver passes high level solutions to the Policy Maker which compiles and sends flow table updates to SDN controller.

This expression is given below.

$$\min \quad \alpha_0 \left(\sum_{(i,j)\in\mathcal{L}} \mu_{i,j}^2 + \sum_{r\in\mathcal{R}} \upsilon_r^2 \right) + \alpha_1 \mathcal{M}_l + \alpha_2 \mathcal{M}_r \tag{3.11}$$

3.3.2 SDN Application

The high level flow rules generated by the optimization solver consist of a device path from source to destination for each demand simply stating the device names. This high level solution is given to our SDN application and transformed first to SDN policies and then compiled into individual Openflow switch flow tables. We implement

our SDN Application using Frenetic, a high level SDN programming language [43]. Frenetic provides a declarative query language for traffic classification and filtering as well as a reactive network policy management library [43]. Frenetic maintains both high level policies as well as low level Openflow forwarding rules. Frenetic is a compositional SDN controller allowing us to combine multiple routing policies with logical expressions using either parallel or sequential composition. The Frenetic Run Time compiling process ensures correct application of our combined policies. This is in contrast to non-compositional SDN applications that parse routing policies in a pipelined manner resulting in complications if flow definitions aren't cleverly arranged [37].

In the SDN application we implement layer 3 routing by using the Frenetic provided source and destination ip address filters to build the port forwarding policies. We use default deny so only the packets that match the installed routing policies will be forwarded in the network, others will be dropped. For evaluation we combined our Gos QoS module with SPa shortest path routing module [108] to handle emerging demand between Gos solutions.

3.4 Long Term Flow Evaluation

The initial evaluation section considers all demand as long term, meaning once a demand is introduced it remains constant for the duration of the test. In a later section we add a more dynamic demand model where combine both short term and long term demand on the network.

3.4.1 Benchmark Generation

To evaluate our framework we created a *benchmark generator* that creates a JSON file containing a detailed network description as well as network demand. The full description of the benchmark are in chapter 1.

3.4.2 Mininet Configuration

We generate the network described in the instance in Mininet [73], recreating the hosts, switches, links. Mininet is an open-source network emulation tool that uses a process based virtualization running all devices network stacks on a single OS kernel. The impact of implementing the actual network stack enables code to be developed and tested in Mininet and ported to other network environments for implementation.

Network devices are implemented as Mininet switches with identifier, port speeds and memory taken from the output of the benchmark generation. Switch capacity, indicates the rate at which a switch can process flows. (It is additionally used in Gos to specify the maximum number of routing rules that can be stored by a switch. In this way, the solution presented by Gos should also be implementable on the switches.) The network infrastructure is generated using the bandwidth capacity provided in the instance. Mininet automatically maps these connections to ports on the switches.

Hosts each have a single network interface and connect to a single edge switch as in the Fat-Tree topology. We use the host identifier provided in the instance and have Mininet auto assign each host a layer 2 MAC and layer 3 IPv4 address. Once the network is generated we build a data structure to maintain a detailed description of the network containing switch port and network addressing assignments from Mininet.

3.4.3 Traffic Generation Model

The routing solution is evaluated in the analysis module by first generating traffic specified in the benchmark. The benchmark generator has an input parameters to indicate the volume of initial demand to generate. The iPerf tool is used to generate traffic for our benchmark in our emulated network for evaluation [1]. iPerf provides the the option to generate traffic for both the TCP and UDP transport layer protocols. We use iPerf in a fixed bandwidth mode for specified time period. We generate simultaneous iPerf commands for each demand in the benchmark. The tool also generates a report that details the results of each test. The result provides several useful measurements for assessment, reported for each second of the test as well as a summary result for the entire test. We use the per second bandwidth measurement in our evaluations to access the quality of each routing configuration. The results of each iPerf test is aggregated across the duration of the test to determine the total traffic on the network at one time.

3.4.4 Experimental Design

As described in the introduction GOSH is a hybrid of two QoS modules, the first simple module routes using shortest path, we call this QoS module SP. The second is Gos which takes all current flows and solves for a global optimum for how to route those flows while minimizing the network resource utilization (switch and link capacity). We perform two major tests:

1. A *static demand* test where all flows are routed using a single QoS module. This allows to compare the relative performance of SP vs. Gos.

2. A *increasing demand* test with three phases, first GOSH routes all flows using Gos in the intermediate as additional demand is placed on the network deploys SP for routing additional demand and finally invokes Gos to solve the new optimum routing, and replaces routing with this new optimum. This test shows the value of the hybrid solution: using traditional QoS mechanisms as flows arrive and the Gos solution can be applied periodically subject to providing enough performance improvement.

Static Demand Bandwidth Test Each test is replicated 10 times and results are provided using an error bar plot with indicators at the min, 10, 25, 50, 75 and 90 percentiles for a 100 second duration. Each individual cycle of a test includes a clean build of all elements including the routing solution, emulation network, the network controller application, flow table install, and the traffic generation.

This test compares the Multi-flow Global solution against an implementation of SP routing using the previously described Fat-tree-4 network with 16 hosts using TCP and varied amounts of traffic. Recall that to increase the amount of demand generated by the demand module we vary the number of flows per host. We generate benchmarks with 32, 64 and 96 flows in this experiment. In each iteration the demand is given to the routing solver (SP , Gos), the routing solution is generated and installed prior the the iPerf commands starting.

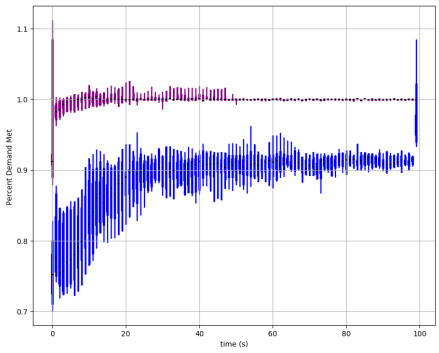
Increasing Demand Bandwidth Test The goal of this test is to better understand the trade-offs of integrating the global solver with increasing demand. In order to facilitate this we use our benchmark generator to generate both initial de-

mand as previously described and additional demand. To generate additional demand our benchmark generator provides an input parameter to determine the volume of additional demand to generate.

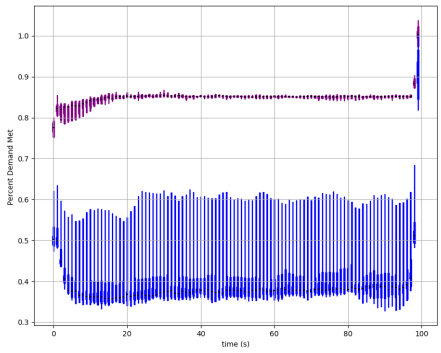
In the test we generate a Fat-tree 4 benchmark, with 16 hosts and 64 flows. We repeat the experiment 10 times and provide the resulting throughput in Figure3.6 . We generate a routing solution for the initial demand using the Gos QoS module in our SDN application. The routing solution is installed in our OpenFlow switches and iPerf commands are generated for the initial demand. We introduce 32 flows of additional demand at the 60 second mark and use the SP module to generate a routing solution. This solution is sent to the Frenetic controller to be recompiled and pushed to the SDN devices. In the final stage of this evaluation we combine the initial and the additional demand and re-optimize using Gos. This high level solution is then send to Frenetic to compile and push updated flow tables to the switches. Looking ahead to our evaluation, we expect to see three periods: 1) initial configuration using Gos 2) handling additional flows using SP and 3) reconfiguration using Gos.

3.5 Long Term Flow Results

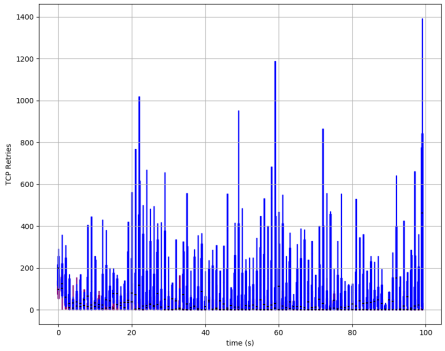
We focus our discussion of results on the two demand tests described in Section 3.4.4. As a reminder, we consider two tests: 1) a *static* test where we compare routing new flows with SP vs optimizing routing new flows using Gos and 2) a *additional demand* test where an initial set of flows is routed with Gos, new arriving flows are routed with SP, and then routing is reconfigured once a solution for all the new flows arrives from Gos. Additionally, we review the solve time of Gos over various instances.



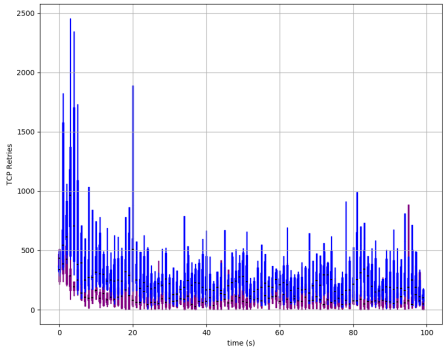
(A) Benchmark A Demand Met



(B) Benchmark B Demand Met

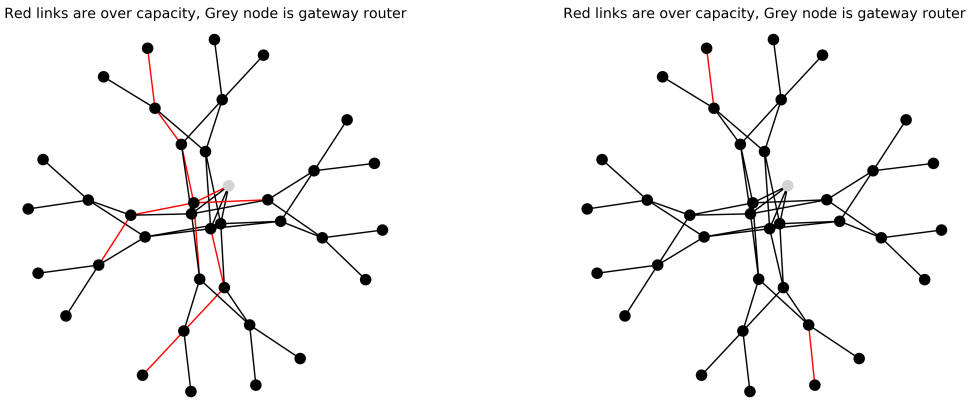


(C) Benchmark A TCP retries



(D) Benchmark B TCP retries

FIGURE 3.4: Shortest Path module in blue and the multi-flow optimization solution routing in purple. Start of bar represents min, widening at 10% and 25% with a line at 50%, reducing width at 75% and 90%, stopping at the max for each. Benchmark A: 98 flows (68 internal and 30 external) totaling 9019 Mbps demand. Benchmark B: 66 flows (46 internal and 20 external) totaling 5719 Mbps demand.



(A) Congestion with Shortest Path Solution

(B) Congestion with Multi-flow optimized solution

FIGURE 3.5: Link utilization diagrams of a static bandwidth test from Figure 3.4a. Edges in red are over capacity.

3.5.1 Static Demand Test

Figure 3.4 displays the results of our static test on two different configurations labled Benchmark A and Benchmark B over 10 runs. Figure 3.4a and Figure 3.4c reference Benchmark A show the demand met and number of TCP retries, and Figure 3.4b and Figure 3.4d show the demand met and number of retries for Benchmark B. As a reminder the data was sent using the iPerf3 tool, generating TCP traffic at a fixed data rate. Each flow demand implemented as an individual iPerf3 command and the aggregate of the results are presented in the figure. Both benchmarks percentage of demand met displayed the most variability at the head and tail of which is partly explained when we consider the TCP protocol's congestion avoidance strategy through additive increases in traffic sending rate with multiplicative back-off. Analysis of the iPerf results substantiated our suspicion as we noted the adjustment of the conges-

tion window, starting lower and generally increasing. The increase in the congestion window is only part of the story, we need to also look at the associated number of TCP retires in Figure 3.4c-d.

In looking at Benchmark A, we notice that GOS performs very well in meeting the demand and that there is very little variability in the results. SP on the other hand has a longer initial period of adjustment where the TCP protocol is adjusting until we see a significant increase in the packets being dropped at about 20 seconds for SP in blue in. We can see the associated -leveling off- of the percentage of demand met at this point, although the the variability continues with the transport layer trying to handle the demand. In looking at the results on Benchmark B we see that GOS still outperforms SP although neither was able to meet all the demand due to the fact that we exceeded the maximum bisection bandwidth of this configuration by over 15%. Again the significant variability in the demand met is closely paired with the number of retires that TCP is sending. These results indicate the importance of balancing demand and resources with multi-flow route optimization to avoid the exponential back-off of the transport layer protocols. These results Figure 3.4 show us the performance in the percentage of demand met and and indicators of good and poor performance in the number of TCP retries but to understand why the retries are necessary we need to understand the network congestion.

For our discussion we will consider a link to be congested if the amount of data being sent on it is over it is capacity. Figure 3.5 shows the congestion of the two solutions SP and GOS for Benchmark A. Each solution was required to route all demand and that neither solution avoided oversubscribing the network. In order to evaluate the solution we look to where this congestion takes place and if it was avoidable. In looking at where the congestion takes place the SP has links congested

at all levels of the network, host-switch and switch-switch while the GOS solution only has congestion at the host-switch edges. Considering the last hop host-switch edges, this congestion is unavoidable given the requirement to route all demand. The switch-switch congestion in this case an indicator of the quality of the solution. With that in mind it is interesting to see where each QoS module suffers from congestion, the SPModule utilizes a set path regardless of the utilization and as a result several links are congested. Conversely the multi-flow optimization solution has balanced traffic and only shows over utilization of unavoidable links.

3.5.2 Increasing Demand Test

The results of this test show the promise of the Hybrid approach of combining a very fast single-flow solution with a globally optimal multi-flow solution. The results are shown in Figure 3.6 show three distinct phases of throughput resulting from the implementation of the 3 separate routing configurations of the same network topology. In the initial phase there is a total of 5184 Mbps demand which is solved with GOS, we can see that the we are meeting all demand (with small variations). In examining the individual link statistics for this segment there are 2 links above capacity both are from switch-host and therefore unavoidable.

In the second phase which begins at the 30 sec mark, we introduced an additional 2412 Mbps of traffic for a total of 7596 Mbps demand. This additional demand is first routed using the SP routing solution and as shown in the results effectively routes and additional 1066 Mbps or 44% of the additional demand. While the SP solution is routing the additional demand, GOS is working on an optimal solution for the combined demand. In is test the average time for GOS to solve the model with an

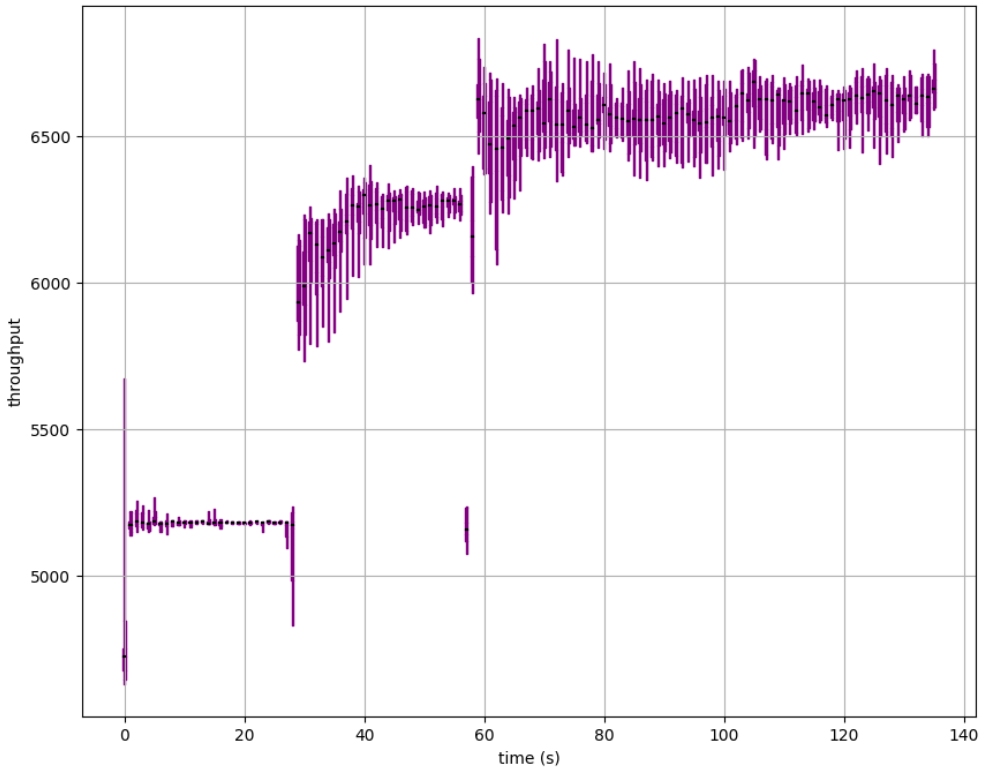


FIGURE 3.6: Results of the additional bandwidth test. Initial demand is routed with multi-flow optimization solution. At 30 seconds additional demand is introduced and routed with the SPmodule. At 60 seconds the combined demand re-optimized and routing is updated with multi-flow solution.

globally optimal solution was under 30 seconds.

In the final phase of this test we pass all the demand to Gos and the routing solution is implemented at 60 second mark. We see an improvement in the throughput at this stage, with the Gos solution able to route 14% more of the additional demand than the SP solution.

3.5.3 Gos Solve Time

Table 3.1 shows the average solve time for GOS over a several instances where both pod size and the number of flows per host were varied. The average solve time is extremely fast through the entirety of the pod 4 instances, but can experience extreme slowdowns starting with the pod 6 networks (several of the pod 6 instances exceeded the time limit of 900 seconds). Surprisingly, almost all of the pod 8 instances did not time out, with most solving in under a handful of minutes. Lastly, just about every pod 10 instance exceeded the 900 second time limit.

There are a couple takeaways from these initial results. First, both the number of network devices and the number of flows greatly impact the model solve time, with flows seemingly having a more significant impact. This is to be expected, however, as the model size (number of variables and constraints) increases proportionally with the number of network devices and the number of flows to be routed. Second, the structure of the flows to be routed seems to also have a very significant impact on the solve time. This is seen most prominently when comparing the high number of time-outs over pod 6 instances with the very few timeouts over pod 8 instances. Perhaps some of the prescribed demands result in many solutions with similar objective lower bounds, resulting in more of the configuration search space having to be explored.

Network Topology				Network Traffic and Solve Time					
Pod	Hosts	Switches	Links	Flows	Avg Solve Time (s)*	% Timed Out	Flows	Avg Solve Time (s)	% Timed Out
4	16	21	52	34	0.67	0	98	5.86	0
6	54	46	171	110	46.1	11	326	24.6	56
8	128	81	400	258	269	11	770	405	11
10	250	126	775	502	755	78	1502	N/A	100

TABLE 3.1: Average solve time for GOS over instances with varying numbers of hosts and flows per host (9 instances per variation). *Solve time is average for instances that did not time out.

Further analysis must be conducted to narrow in on a more concrete explanation for this phenomenon.

3.6 Dynamic Flow Evaluation

Using the previously discussed evaluation process as a baseline I would like to explore modifications to the network demand and traffic generation. In this section we add the notion of short and long term demand or traffic.

3.6.1 Benchmark Generation

The instance generator was modified to categorize each network demand as either short or long. The short designation is intended to represent bursty traffic such as a web page request. The long designation identifies traffic that requires a minimum bandwidth to achieve the desired QoS as in VoIP calls or video streams.

3.6.2 Mininet Configuration

-no change

3.6.3 Traffic Generation

In this set of tests we have 90% of all flows small at 10 Mbps long duration flow or 10 MB short duration flow. We generate 10% large flows 500 Mbps for long duration traffic or 500 MB for short duration traffic. In this model we leverage the short and long designation of each demand. We introduce an additional short demand traffic module in which a Poission distribution is used to determine the inter arrival time of each flow, centered at a half-second. The long duration traffic is initiated at the start of each test and continued for the duration of the test. In this traffic model we focus on the performance the long duration traffic with a given routing configuration. The short flows that are processed with the Poission distribution are considered background traffic and therefore not included in the aggregated iPerf output. However, even though the short flows are not included in our analysis they are being generated and routed on the network during the test, contending for network resources with the QoS traffic.

3.6.4 Additional Routing Modules

We develop two additional routing modules to be used in the evaluation process.

1. DWSP *Dijkstra's Weighted Shortest Path* is designed as a global solver using the bandwidth requirement for each demand to weight the edges. A random selection process is used to order the processing of individual demands through the algorithm. The graph edge weights are updated after each pass of the algorithm. We use DWSP to compare against Gos as a comparison of global solvers, specifically looking at network congestion that results from each solution.

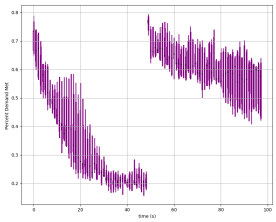
2. LBSP *Load Balanced Shortest Path* is implemented as an intermediate between SP and DWSP. The algorithm is implemented exactly as DWSP with the exception in the weighting of the edges. Here as demands are assigned to edges the weight of each edge is incremented, rather than using the bandwidth requirement. This has the affect of load balancing, taking advantage of the multitude of equal length paths available in the Fat-Tree topology. This module is implemented as a single flow solver paired with Gos (not against) in our evaluation.

3.6.5 Dynamic Flow Experimental Design

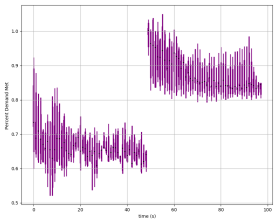
Dynamic Traffic Test The Integration of LBSP as the single flow solver with Gos as a multi-flow solver for long duration traffic. In stage one we route all traffic with LBSP. In stage two we pass all long duration traffic traffic to Gos and the short duration traffic to LBSP. The long duration traffic is solved first with the resulting edge utilization passed to the single flow solver LBSP as a primer.

In the test we generate several instances of two classes of benchmarks using our instance generator. Benchmark class C are all Fat-Tree-4 with 66 demands about half short and half long duration flows. When the traffic is generated for this class of benchmark there are 35 long duration flows that run for the duration of the test and over 1200 short duration flows, about 10 per second arrive. In the benchmark D class we have several instances of Fat-tree 4 with 96 demand pairs, again split half short and half long with the same inter-arrival time distribution between flows.

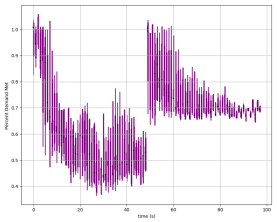
We repeat each experiment 5 times and provide the resulting percentage of demand met for the long duration flows in Figure 3.9 and Figure 3.7 . First we generate a



(A) random seed 6



(B) random seed 7



(C) random seed 8

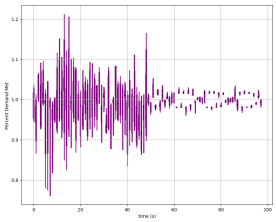
FIGURE 3.7: Benchmark class C Percent Demand Met: LBSP routing 0-50 seconds then a combination GOS and LBSP routing 50-100 seconds. There are 35 long duration flows and over 1200 short flows. Start of bar represents min, widening at 10% and 25% with a line at 50%, reducing width at 75% and 90%, stopping at the max for each.

routing solution for all the demand (long and short) using the LBSP routing module in our SDN application. The routing solution is installed in our OpenFlow switches and iPerf commands are generated as previously described. While these flows are being routed we run GOS on the long duration demand and then re-calculate the routes for the short duration demand with LBSP. We push the update solution to the SDN devices at the halfway mark of each test and continue the iPerf traffic generation process for the second half.

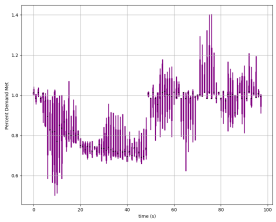
3.7 Dynamic Flow Results

3.7.1 Dynamic Flow Test

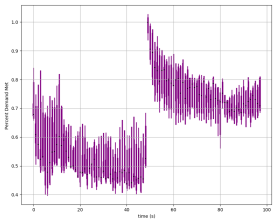
The results show that using a hybrid approach to handle different traffic demand patterns shows promise. The multi-flow solution determines the optimal routes for our long duration traffic while we pass the short durration, bursty traffic off to a



(A) random seed 3



(B) random seed 4



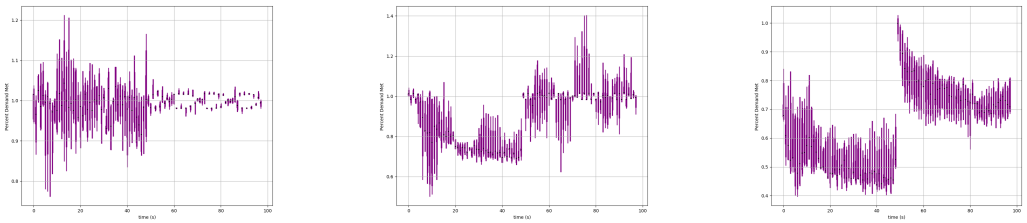
(C) random seed 5

FIGURE 3.8: Benchmark class D Percent Demand Met: LBSP routing 0-50 seconds then a combination GOS and LBSP routing 50-100 seconds. There are 50 long duration flows and over 1200 short flows. Start of bar represents min, widening at 10% and 25% with a line at 50%, reducing width at 75% and 90%, stopping at the max for each.

single-flow solver. The results are shown in Figure 3.7 and Figure 3.9. As a note, in places where the graph indicates that over 100% of demand is being met results from TCP retransmissions from previous time periods.

The results show two distinct phases 0-50 seconds using only a LBSP routing solutions and 50-100 seconds using a combination of GOS and LBSP solutions. The graphs clearly reflect where the updated routing configurations are pushed at the mid point normally resulting in a higher percentage of demand met or less variance. This step or lowering of variance is consistent across most of our results of various benchmarks. Although not shown here there have been a low percentage of instances in testing that showed a slight decrease in the hybrid routing which we attribute to the placement and volume of the bursty traffic. Recall, that in our testing we route all demand even if the volume of traffic will oversubscribe a portion of the network. Additionally, we notice that in several of the results there is a tendency in the later stages of the test to see a downward tail as a result of the TCP congestion avoidance technique. As the routing of traffic progresses with each phase the volume of bursty

traffic introduces congestion the the long term traffic is eventually impacted.



(A) random seed 3 (B) random seed 4 (C) random seed 5

FIGURE 3.9: Benchmark class D Percent Demand Met: LBSP routing 0-50 seconds then a combination GOS and LBSP routing 50-100 seconds. There are 50 long duration flows and over 1200 short flows. Start of bar represents min, widening at 10% and 25% with a line at 50%, reducing width at 75% and 90%, stopping at the max for each.

3.8 Conclusion

The hybrid implementation of a single flow solver to handle dynamic demand with a multi-flow solver to re-establish network resource optimization shows promise. The performance of the global solution is better than the shortest path routing solution in terms of throughput. The apparent trade-off is the time it takes to solve the Global model compared to the stability of the demand. The solution time is a factor of the size of the network and the number of flows needed to be steered through the network. The global solution's performance improvements are only achieved when the traffic remains stable long enough to solve and implement the improved routing.

Chapter 4

Functionality and Security - Layered Framework

4.1 Introduction

Network configuration is a crucial task in any enterprise. Administrators balance functionality, performance, security, cost, and other industry specific requirements. The resulting configuration is subject to periodic analysis and redesign due to red team recommendations, emerging threats, and changing priorities.

Our Contribution This work introduces a new optimization framework that finds network configurations that satisfy multiple (conflicting) requirements. We focus on data center networks (DCN) that use software defined networking (SDN). Background on these settings is in Chapter 1. Our framework is called DocSDN (Dynamic and Optimal Configuration of Software-Defined Networks).

DocSDN searches for network configurations that simultaneously satisfy multiple properties. DocSDN is organized into layers that consider different properties. The core of DocSDN is a multistage optimization that decouples search on "orthogonal" concerns. The majority of the technical work is to effectively separate concerns so the optimization problems remain tractable. Our framework is designed to continually produce network configurations based on changing requirements and threats. It frees IT personnel from the complex question of how to satisfy multiple requirements and can quickly incorporate new threat information.

DocSDN focuses on achieving functional requirements (such as network reachability and flow satisfaction) and limiting security risk (such as isolating high risk nodes and nodes under denial of service attack). Naturally, other layers such as performance or cost can be incorporated. The search for a good configuration could be organized in many ways. State-of-the-art approaches assess different properties in isolation, frustrating search for a solution that satisfies all requirements. Ideally, a framework should search for a configuration that simultaneously satisfies all requirements. This extreme is unlikely to be tractable on all but the smallest networks. DocSDN mediates between these approaches separating the functional and security search problems but introducing a feedback loop between the two search problems based on *cuts*.

In the proposed organization the functional layer is "above" the security layer. Through the feedback loop, the security layer describes a problematic part of the network to the functional layer. The functional layer then refines its model and searches for a functional configuration that satisfies an additional *constraint*. This has the effect of blocking the problematic part of the configuration. Currently, the feedback signal is a pair of nodes that should not be proximate in the network. After

multiple iterations the two layers jointly produce a solution that optimizes the SDN configuration both with respect to functionality and security risks.

DocSDN provides solutions of improving quality before the final solution. Thus, the network can be reconfigured once the objective improves on the current configuration by a large enough amount (to justify the cost/impact of reconfiguration).

The underlying optimization problems are NP-hard but optimization technology has seen tremendous advances in performance during the past few decades. Since 1991, mathematical programming solvers have delivered speedups of 11 orders of magnitude [20, 45]. Hybrid techniques such as Benders decomposition [16, 30, 52, 53] and column generation [12, 49, 72] (aka, Dantzig-Wolfe decomposition [34]) made it possible to solve huge problems thanks to on-demand generation of macroscopic variables and the dynamic addition of critical constraints. Large Neighborhood Search [111] further contributed to delivering high-quality solution within constrained time budgets.

These techniques are beginning to see adoption in network security. Yu et al. recently applied stochastic optimization with Bender's decomposition to assess network risk under uncertainty for IoT devices [126]. They used Bender's decomposition on a scenario-based stochastic optimization model to produce a parent problem that chooses a deployment plan while children are concerned with *choosing* the optimal nodes to serve the demands in individual scenarios. In comparison, our approach addresses *both functional and security requirements*. It relies on Bender's cuts from the security layer (child) to rule out vulnerable functional plans whose routing paths fail to adequately minimize risks and maximize served clients. We now briefly describe the framework (a formal description is in Section 4.3) and present an illustrative example.

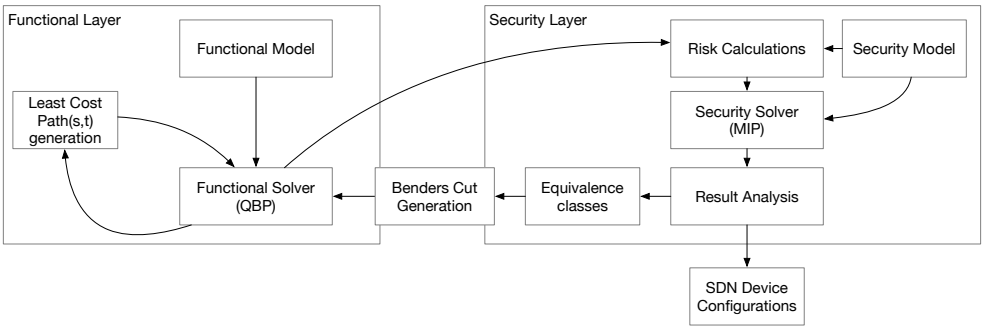


FIGURE 4.1: DOCSDN Framework. A layered decomposition that breaks down configuration synthesis into functional and security layers.

Overview of DocSDN Figure 4.1 presents an overview of the framework. The functional layer takes as input a *Functional Model* that describes the network including the physical topology, capacity, the allowable communication patterns and the demand requirements. Network reachability begins with a priming procedure that generates the k-least cost paths to the optimizer for each source/destination pair in the demand requirements. The objective for the functional layer is to find a logical topology (a collection of routed paths) that meets all demand requirements while favoring shorter length routing paths and load balancing. The program is formulated as quadratic binary program (QBP). The solution as determined by the functional layer is passed to the security layer.

The output of the functional layer and a *security model* are the input for the security layer. The current configuration is fed to a module that uses risk assessments for the individual network devices (obtained for example using a vulnerability database) to assess the overall risk of the entire configuration. In our current implementation this risk calculation is based on a simple risk propagation model where a path's risk is based on the risk of nodes on the path and close to the source and sink. The security layer can deploy firewalls and deep packet inspection as network defenses. Since these mechanisms affect route capacity, the security layer has a dual objective function: 1)

maximizing the functional objective and 2) minimizing security risk. The security objective is formulated as a mixed integer program (*MIP*). When the security search completes, it proposes nodes to the functional layer that should be separated. As an example, a high value node with low risk may be placed in a different (virtual) LAN than a high risk node. These *Benders cuts* are designed to entice a better logical topology from a subsequent iteration in the functional layer. This feedback loop between the two layers can iterate multiple times. When no further cuts are available, the overall output is a set of configuration rules.

An example configuration This section describes an application of our framework to automatically respond to a distributed denial of service (DDoS) attack. Current DDoS attacks demonstrate peak volume of 1 Tbps [68]. Many DDoS defense techniques require changes to the network behavior by rate limiting, filtering, or reconfiguring the network (see [57,58,85,95,128]). Recent techniques [40] leverage SDNs to react to DDoS attacks in a dynamic and flexible manner. We show how such a response would work in our framework using a toy network illustrated in Figure 4.2. A more realistic network and the framework's response are described in Section 5.4. We stress that DDoS attacks are often short in timescale making human diagnosis and reaction costly or impractical. Consider a focused DDoS attack against a number of services in an enterprise but not the entirety of its publicly accessible address space. (The Great Cannon's attack against GreatFire targeted two specific Github repositories [82].) We assume a service hosted by H_1 is targeted, while services on H_2, H_3 and H_4 are not.

Recall, the functional layer establishes a logical topology (forwarding rules) while

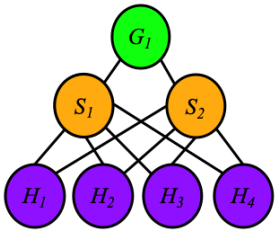


FIGURE 4.2: Toy network example with a single gateway device G_1, two intermediate switches S_1 and S_2 and four hosts. We assume the switches are physically connected to all hosts.

the security layer adds network defenses (packet inspection modules and firewall rules). We elide how the attack is detected and assume it increases the risk score for H_1 in the security model.

The first iteration The functional layer proposes a candidate configuration where G_1 routes all traffic intended for H_1 and H_2 to S_1 which then forwards the traffic and G_1 routes traffic intended for H_3 and H_4 to S_2 which then forwards the traffic. This is the first candidate solution presented to the security layer.

Since H_1 is high risk the security layer proposes a firewall at S_1 to block all port 80 traffic. This reduces risk at the cost of blocking all traffic to H_2. Of course, in real firewalls more fine-grained rules are possible, this simplified example is meant to illustrate a case where collateral damage to the functional objective is necessary to achieve the security objective. Since traffic is being blocked to a node with low risk, the security layer asks the functional layer to separate H_1 and H_2 so H_2 does not suffer.

Repeated iterations The functional layer now has a constraint that H_1 and H_2 should not be collocated in the network. As such, it proposes a new configuration with H_1 and H_3 under S_1 and H_2 and H_4 under S_2. This is then sent to the security layer. The security layer makes a similar assessment and proposes a firewall rule at

S_2, finds this recommendation hurts functionality and requests separation of H_1 and H_3.

This process repeats with the functional layer proposing to collocate H_1 and H_4. The security layer similarly asks to separate H_1 and H_4. Finally, H_1 is segregated from all other nodes. This produces a configuration where H_1 is the only child of S_1. Note that having H_2, H_3 and H_4 under a single switch may hurt performance but the effect is less than blocking traffic to one of the nodes entirely. DocSDN can then output the candidate solution as high level SDN fragments (using a high-level language like Frenetic [44]).

Recovery Importantly, when the DDoS abates, DocSDN automatically reruns with a changed risk for H_1, outputting a binary tree.

Organization The rest of the work is organized as follows: Section 4.2 provides background on our application and discusses related work, Section 4.3 describes our framework and accompanying optimization models, Section 5.4 evaluates the framework and finally Section 5.5 concludes.

4.2 Background and Related Work

Our framework is intended to be modular and allow integration of prior work on evaluating network configurations. As such there is a breadth of relevant work. Due to space constraints we focus on the most relevant works. In the conclusion we elaborate on the characteristics needed to integrate a prior assessment tool into our framework (see Section 5.5).

Measuring Network Risk Known threats against computer systems are main-

tained by governments and industry. Common Vulnerabilities and Exposures (CVE) is a publicly available dictionary including an identifier and description of known vulnerabilities [32], CVE does not provide a severity score or priority ranking for vulnerabilities. The US National Vulnerability Database (NVD) [92] is provided by the US National Institute of Standards and Technology (NIST). The NVD augments the CVE, adding severity scores and impact ratings for vulnerabilities in the CVE.

There are many mechanisms for measuring the security risk on a network [27, 59, 80, 115, 116]. Lippmann et al. present a network security model which computes risk based on a list of the most current threats [79]. This model implements a cycle of observe network state, compute risk, prioritize risk, and mitigate the risk.

This loop is often codified into an *attack graph* [56, 62, 106]. Attack graphs try to model the most likely paths that an attacker could use to penetrate a network. Attack Graphs often leverage one or more of the aforementioned vulnerability assessment tools as input, combined with a network topology and device software configurations to generate the graph. Current attack graph technologies provide recommendations to network administrators that effectively remove edges from the graph and trigger a re-evaluation of the utility for the attacker. To the best of our knowledge, current practice does not leverage network risk measurement into constraints used for the generation of new configurations.

Network Reachability The expansion of SDN has aided the applicability of formal verification to computer networks. Prior to SDN, the lack of clear separation between the data and control plain created an intractable problem when considering a network of any scale. Bounded model checking using SAT and SMT solvers [13, 129] can currently verify reachability properties in networks with several thousands of nodes.

Configuration Search Constraint Programming (CP) was introduced in the late 1980s [104] and is used for scheduling [11], routing, and configuration problems. Large-scale optimization problems are often decomposed including Benders [30] and Dantzig-Wolfe [34]. *Soft constraints* or Lagrangian relaxation are used for over-constrained problems or when the problem is too computationally expensive. Stochastic optimization techniques have been used for many applications in resilience [22, 88] and the underlying methodologies are a key part of this research. Prior work in configuration management with constraint programming [29, 75] focused on connectivity or security. We are not aware of any work that balances these two objectives in a meaningful way.

4.3 Implementation

Figure 4.1 outlines the overall structure of the DocSDN framework. layer interconnections as well as their internals. The functional layer uses a mathematical optimization model that is fed to a quadratic mixed boolean programming (QBP) solver alongside an initial set of least-cost paths to be considered to service the required flows. The security layer receives the topology chosen by the functional layer and a security model to solve, with a mixed-integer programming (MIP) solver, the risk minimization problem. The output can result in low-risk flows being blocked as a consequence of deploying firewalls to mitigate high-risk flows. A result analysis module then produces *equivalence classes* that are sent back to the functional layer to request the separation of specific flows that should not share paths, with the goal of minimizing the collateral damage to low-risk flows. These equivalence classes gen-

erate additional constraints, known as Bender's cuts, that are added to the functional solver for a new iteration. The remainder of this section describes the major modules in Figure 4.1.

4.3.1 Functional Layer

The mathematical optimization model in the functional layer is a quadratic mixed binary programming model. In constraint programming the four main components are Inputs, Variables, Constraints, and an Objective function. Inputs are below.

Inputs

$\mathcal{N}$ – the set of all network devices

$\mathcal{E}$ – the set of edges (pair of vertices) connecting network devices

$\mathcal{T}$ – the set of types of traffic to be routed

$\mathcal{F}$ – the set of $(s, t, T) \in \mathcal{N} \times \mathcal{N} \times \mathcal{T}$ tuples defining desired traffic flows of type T from source node s to sink node t.

$D(f) : \mathcal{F} \to \mathbb{R}$ – the actual demand for each flow $f \in \mathcal{F}$

$\mathcal{C} \subseteq 2^{\mathcal{N}}$ – a subset of sets of network devices

$\mathcal{R} \subseteq \mathcal{C} \times \mathcal{C}$ – pairs (c_1, c_2) of equivalence classes that segregate traffic from c_1 to c_2.

$\mathcal{P}$ – the set of all paths

$P(e) : \mathcal{E} \to \mathcal{P}$ – the set of all paths containing edge e

$P(n) : \mathcal{N} \to \mathcal{P}$ – the set of all paths containing node n

$P(c) : \mathcal{C} \to \mathcal{P}$ – the set of all paths containing a node in c

$N(p) : \mathcal{P} \to \mathcal{C}$ – the set of nodes appearing in path p

$P(s, t) : \mathcal{N} \times \mathcal{N} \to \mathcal{P}$ – the set of all paths $s \to t$

$cap(e) : \mathcal{E} \to \mathbb{R}$ – gives the capacity of an edge e.

Variables

$active_{p,T} \in \{0,1\}$, – for every path $p \in \mathcal{P}$ and traffic type $T \in \mathcal{T}$, indicates whether path p carries traffic of type T

$flow_{p,T} \in \mathbb{R}_{\geq 0}$ – for every path $p \in \mathcal{P}$ and traffic type $T \in \mathcal{T}$, amount of flow of type T that is sent along path p

$equiv_{c,n} \in \{0,1\}$ – does node $n \in \mathcal{N}$ appear in an active path together with a node in equivalence class c

$share_{c_1,c_2,n} \in \{0,1\}$ – indicates whether node $n \in \mathcal{N}$ appears on any active path with nodes in classes $c_1, c_2 \in \mathcal{C}$. Namely,

$$share_{c_1,c_2,n} \Leftrightarrow n \in \left(\left(\cup_{p \in P(c_1): active_{p,*}} N(p)\right) \cap \left(\cup_{p \in P(c_2): active_{p,*}} N(p)\right) \right)$$

$active_{p,*} = 1$ if there is a type $T \in \mathcal{T}$ where $active_{p,T} = 1$

$load_n \in \mathbb{R}$ – the amount of flow that goes through node n

$loadObj$ – the sum of squares of all $load_n$ variables.

Constraints

$$\sum_{p \in P(e), T \in \mathcal{T}} flow_{p,T} \leq cap(e), \ \ \forall e \in \mathcal{E} \tag{4.1}$$

$$\sum_{p \in P(s,t)} flow_{p,T} \geq D(s,t,T), \ \ \forall (s,t,T) \in \mathcal{F} \tag{4.2}$$

$$active_{p,T} = 1 \rightarrow flow_{p,T} \geq 1, \forall (s,t,T) \in \mathcal{F}, p \in P(s,t) \tag{4.3}$$

$$\sum_{p \in P(s,t)} active_{p,T} = 1, \ \ \forall (s,t,T) \in \mathcal{F} \tag{4.4}$$

$$equiv_{c,n} = \bigvee_{p \in P(n) \cap P(c)} (active_{p,T}), \forall T \in \mathcal{T}, n \in \mathcal{N}, c \in \mathcal{C} \tag{4.5}$$

$$share_{c_1,c_2,n} = equiv_{c_1,n} \wedge equiv_{c_2,n}, \forall n \in \mathcal{N}, (c_1, c_2) \in \mathcal{R} \tag{4.6}$$

$$load_n = \sum_{p \in P(n), T \in \mathcal{T}} flow_{p,T}, \forall n \in \mathcal{N} \tag{4.7}$$

Equation 4.1 enforces the edge capacity constraint to service the demand of all paths flowing through it. Equation 4.2 ensures that enough capacity is available to meet the demand of an (s,t,T) flow. Equation 4.3 ensures that some non-zero capacity is used if a specific path is activated (conversely, an inactive path can only have a 0 flow). Equation 4.4 states that a single path should be chosen to service a given flow $f \in \mathcal{F}$. Equations 4.5 define the auxiliary variables $equiv_{c,n}$ as true if and only if node $n \in \mathcal{N}$ appears on an active path sharing a node with the equivalence class $c \in \mathcal{C}$. Equation 4.6 defines an active path that shares at least one node with two classes. Finally, equation 4.7 defines the load of a node as the sum of the flows associated to active paths passing through node n.

Objective

$$\min \begin{pmatrix} \alpha_0 & \sum_{p,T} len(p) * flow_{p,T} & + \\ \alpha_1 & \sum_{(c_1,c_2) \in \mathcal{R}, n \in \mathcal{N}} (share_{c_1,c_2,n} - 1) & + \\ \alpha_2 & \sum_{n \in \mathcal{N}} (load_n)^2 & \end{pmatrix} \tag{4.8}$$

The objective function 4.8 in this model is a weighted sum of three terms. The first term captures the total flows which are penalized by the length of the path used to

dispatch those flows (such policies are codified in OSFP [87] and BGP practice [48]). The second term gives a unit credit each time equivalence classes on the segregation list $\mathcal{R}$ do not share a node. (Due to this term, the objective value of the final solution may change between iterations of the functional layer.) The third and final term contribute to a bias towards solutions that achieve load balancing thanks to the quadratic component which heavily penalizes nodes with large loads.

Solving the Functional Model The functional model starts with empty sets $\mathcal{C}$ and $\mathcal{R}$ which are augmented with each iteration of the framework. New sets of nodes are added to $\mathcal{C}$ and new segregation rules are added to $\mathcal{R}$ (by the security layer). In the current implementation, least cost paths between pairs of nodes s, t are not generated "on demand". Instead, the generation is limited to the first best k such paths, for increasing values of k. This process will ultimately be improved to use column generation techniques [34].

4.3.2 Risk Calculation

After the functional layer finds an optimal solution, it passes this solution to the risk calculation procedure. This input is the set of active paths. This module calculates the effective risk to the network for each path and traffic type.

Inputs

$risk(n, T) : \mathcal{N} \times \mathcal{T} \rightarrow \mathbb{R}$ – the risk inherit to network device n for traffic of type T ($risk(n, T) \geq 1$)

$d_k(n) : \mathcal{N} \rightarrow 2^{\mathcal{N}}$ – the set of nodes at a distance at most k from n in the logical topology

Calculation

Given an *active* path $p \in \mathcal{P}$ with source s and sink t, the calculation proceeds by partitioning the set of nodes of the path into three segments: the nodes "close" to the source s, "close" to the sink t and the nodes "in between". Closeness is characterized by the function d_k and is meant to capture any connected node over the logical topology which sits no more that k hops away. Given this partition, $flowRisk(p, T)$ is:

$$\begin{aligned} flowRisk(p,T) &= \sum_{i \in d_2(s) \cup d_2(t)} risk(i,T)^2 + \\ & \quad \sum_{i \in N(p) \setminus (d_2(s) \cup d_2(t))} risk(i,T)^2 \end{aligned}$$

We use $k = 2$ to model nodes on the same LAN. The rationale is to impart to source s and sink t risk resulting from *lateral movement* of attacks. All other nodes contribute to the overall path risk in proportion to the square of their own risks. We expect in most networks for $d_2(s)$ and $d_2(t)$ to include nodes not directly on the path (like nodes on the same LAN). The input path risk calculation $flowRisk(p, T)$ is modular and can be augmented using other risk calculation methods.

4.3.3 Security Layer

The mathematical optimization model in the security layer is a mixed integer programming model. We similarly present the inputs, variables, constraints, and objective for the security layer. Its inputs are given below. Also note that all the variables from the functional model are *constants*.

Inputs

$mem(n) : \mathcal{N} \to \mathbb{R}$ – the memory resources of SDN device n

$fwCost(T) : \mathcal{T} \to \mathbb{R}$ – the memory footprint for a firewall blocking traffic type T

$piCost$ – the memory footprint for a packet inspection post

$fwComp$ – the complexity footprint for adding a firewall

$piComp$ – the complexity footprint for adding a packet inspection post to the network

$penalty(p, T) : \mathcal{P} \times \mathcal{T} \to \mathbb{R}$ – the penalty for blocking a unit of flow of type T along path p

$rank(n, p) : \mathcal{N} \times \mathcal{P} \to \mathbb{Z}$ – the position of node n in path p

$flowRisk(p, T) : \mathcal{P} \times \mathcal{T} \to \mathbb{R}_{\geq 0}$ – above risk calculation

Variables

$fw_{n,T} \in \{0, 1\}$ – does a firewall block traffic type T at n

$pi_n \in \{0, 1\}$ – is there packet inspection at network device n

$fwOR_{n,T} \in \{0, 1\}$ – is there a *block everything* or *block traffic of type* T firewall at network device n

$fwOP_{p,T} \in \{0, 1\}$ – is there a firewall on path p

$rf_{p,T} \in [0, 1]$ – risk factor for path $p \in P(s, t)$ servicing flow $(s, t, T) \in \mathcal{F}$

$RMfw_{p,n,T} \in [0, 1]$ – used in the riskFactor calculation

$RMpi_{p,n,T} \in [0, 1]$ – used in the riskFactor calculation

Constraints

$$fwOR_{n,T} = fw_{n,T} \vee fw_{n,*}, \forall n \in \mathcal{N}, T \in \mathcal{T} \tag{4.9}$$

$$\sum_{T \in \mathcal{T} \cup \{*\}} fwCost_T \cdot fw_{n,T} + piCost \cdot pi_n \leq mem_n, \forall n \in \mathcal{N} \tag{4.10}$$

$$fwOP_{p,T} = \bigvee_{n \in N(p)} (fwOR_{n,T}), \forall T \in \mathcal{T}, p \in \mathcal{P} : active_{p,T} \tag{4.11}$$

$$RMfw_{p,n,T} = 1 - (.5)^{rank(n,p)} \cdot fwOR_{n,T}, \quad \forall p \in P(s,t), n \in N(p), (s,t,T) \in \mathcal{F} \tag{4.12}$$

$$RMpi_{p,n,T} = 1 - 0.1 \cdot (.5)^{rank(n,p)} \cdot pi_n, \quad \forall p \in P(s,t), n \in N(p), (s,t,T) \in \mathcal{F} \tag{4.13}$$

$$rf_{p,T} = \min \bigcup_{n \in N(p)} \{RMfw_{p,n,T}, RMpi_{p,n,T}\}, \quad \forall T \in \mathcal{T}, p \in \mathcal{P} : active_{p,T} \tag{4.14}$$

Equation 4.9 is used to define the presence of a firewall that will block traffic of type T at a node n. Equation 4.10 ensures that the memory footprint in SDN node n for the deployment of the firewall and the packet inspection logic does not exceed the device memory. Equation 4.11 links the presence of a firewall that will block traffic of type T on a path with the presence of a firewall that will block traffic of type T on any node along the active path. Equation 4.12 defines the minimum risk factor associated to a firewall. The earlier on the path the firewall is deployed, the lower the risk. Equation 4.13 similarly defines the minimal risk. Equation 4.14 defines the composite risk factor.

Objective

$$\min \begin{pmatrix} \beta_0 \left(\sum_{n,T} fwComp \cdot fw_{n,T} + \sum_n piComp \cdot pi_n \right) + \\ \beta_1 \sum_n load_n \cdot pi_n + \\ \beta_2 \sum_{p,T} penalty(p,T) \cdot flow_{p,T} \cdot fwOP_{p,T} + \\ \beta_3 \sum_{p,T} flowRisk_{p,T} \cdot rf_{p,T} \end{pmatrix} \tag{4.15}$$

The objective function defined in equation 4.15 is a weighted sum of four distinct

terms that focus on minimizing the network complexity based on security resources deployed, the load induced by inspection posts, the penalties incurred from dropping desirable flows due to firewall placement and finally the residual risk. This model is a classic mixed integer programming formulation.

4.3.4 Result Analysis

The result analysis module tries to generate cuts for the functional layer with the goal of improving both functionality and security. To generate cuts, this module will form equivalence classes of network nodes and pass back certain pairs of these classes, one at a time, to the functional layer. Each pair of classes describes a segregation rule, or a cut, to which to functional layer will adhere to as much as possible.

After the functional and security layers are re-optimized using the most recent cut, the result analysis module determines whether the cut was beneficial or harmful based on the objectives of each layer. If the cut is deemed to have been beneficial, we permanently keep it as a constraint, repopulate the cut queue, and continue the process.

If the cut is deemed to have been harmful, it is removed from the functional layer's constraint pool. Then the next cut in the queue will be passed back to the functional layer. If the cut queue is empty, the feedback mechanism terminates and we output the best solution found.

We note that since this process only provides pairs of nodes it is a heuristic. It may be necessary for many nodes to simultaneously be separated to arrive at a global optimum. This mechanism performed well in our experiments.

4.3.5 Layer Coordination

It is valuable to review how the layers coordinate. The functional layer sends to the security layer a set of paths that implements the routing within the network to serve the specified flows while satisfying a set of segregation requirements. The security layer first computes *risks* for these paths based on its knowledge of the traffic. The paths, their risk and the security model are then tasked with deploying packet inspection apparatus as well as firewalls within that logical topology to monitor the traffic and block threats (risky traffic). Once the security model is solved to optimality, an analysis can determine whether the proposed logical topology is beneficial or not (w.r.t. its objective) and even suggest further equivalence classes for network nodes as well as segregation rules to be sent back to the functional layer for another iteration. Fundamentally, the coordination signal boils down to additional equivalence classes to group nodes together with segregation rules to separate paths that include network nodes in "antagonistic" equivalence classes.

4.3.6 Outputs

When the set of potential cuts is empty, the proposed configuration can be parsed and translated into SDN language fragments to be deployed on the network devices in order to obtain the desired logical network topology put forth by our framework.

Within our sample network, we consider having two main types of devices: switches/routers and hosts. In order to model traffic between internal and external entities we utilize two gateway switches which represent the boundary of our network. For generality we consider two traffic types (A and B) which could represent any type of traffic such as web and storage. We also classify traffic as internal and external, with external traffic

traversing one of the gateways. We allow only half of our hosts to communicate with external sources by allowing them to connect to one of the two gateways. Further, all hosts are involved in internal communications. In this instance we have 16 hosts and we generated 60 flows, 44 of them being internal and 16 being external.

Additionally, our setup simulates an emergent vulnerability/active attack. We select two hosts that are highly vulnerable to, or being targeted on, a specific type of traffic, resulting in a significant increase in their risk for the corresponding type of traffic. In particular, this could represent at DDoS attack on these two hosts.

We run multiple experiments providing the framework increasing numbers of starting paths between source and destination (from the priming procedure) to determine the impact on the solution quality.

Our implementation was built in Python 3.6 using the Gurobi 8.1 optimization library [93]. The experiments were run on a machine running Ubuntu 18.04 and equipped with an Intel Core i9-8950HK processor operating at 2.90 GHz with a 12 MB Cache and 32 GB of physical memory.

4.4 Evaluation/Results

We begin with the model's resolution on the above scenario using the 10 shortest paths per source and destination pair to prime the optimization. We assume equal demand for each of the 60 flows and two high risk hosts (in red in Figure 4.3). In each iteration the functional layer of the framework generates a candidate topology and passes this solution to the security layer. The security layer then calculates the network risk and deploys firewalls (represented with rectangles).

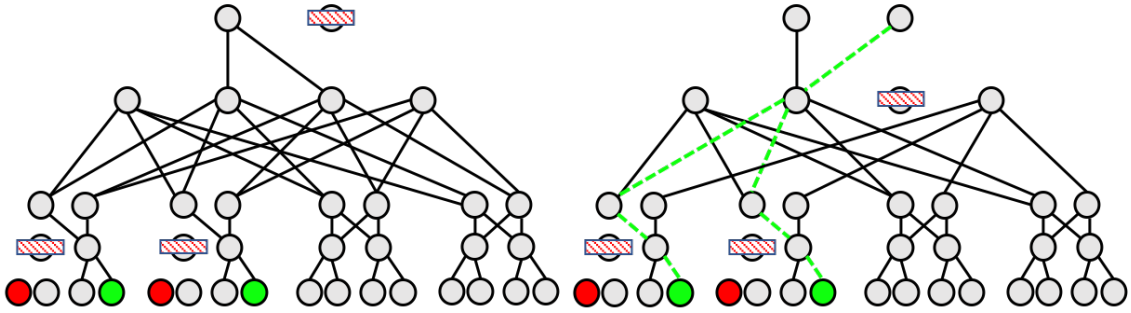


FIGURE 4.3: Illustration of the Fat-Tree network after the first pass through both layers of the framework (left) and the final configuration (right). Note: Firewalls are depicted with rectangles, red nodes represent high risk nodes, green nodes represent nodes that are initially blocked and recovered in the final configuration, utilizing the updated routes shown in dashed lines. The modification of the logical topology allows for more intelligent firewall placement balancing both functionality and security.

In the initial configuration, the gateway on the right serves both low risk (shown in green) and high risk hosts (shown in red). Deploying a firewall on this gateway significantly reduces the network risk and selected as optimal by the security layer. Importantly, this results in collateral damage as flows to low risk hosts are blocked. In total, the first iteration through the framework deploys 3 firewalls which block 12 flows, 8 of which are high risk.

Iteratively, the security layer proposes separation of these collateral nodes from the high-risk nodes. The solution of the last framework iteration (493 candidates cuts are proposed, 40 prove beneficial) is shown on the right. This configuration routes all high risk flows through one core switch where a firewall is now deployed. Meanwhile, all the low risk flows access the gateways through a separate core switch.

Overall we conducted six experiments with this configuration modifying only the number of paths ($\{10, 20, 30, 40, 50, 100\}$) being used to prime the functional layer for each source-destination pair. Ultimately, this process will be dynamic and use column-generation. Complete experimental results are shown in Table 4.1. A few

	10 paths	20 paths	30 paths	40 paths	50 paths	100 paths
Initial Flows Blocked	12	12	12	12	12	12
Final Flows Blocked	8	8	8	8	8	8
Initial Functional Objective	2012	2012	2012	2012	2012	2012
Final Functional Objective	2014	2014	2014	2015	2014	2015
Cut Reward	-8450	-4260	-7210	-4900	-5540	-3840
Initial Security Objective	13735	13724	13729	13723	13696	13726
Final Security Objective	13356	13356	13356	13356	13356	13348
Initial Network Risk	10425	10414	10419	10413	10386	10416
Final Network Risk	11646	11646	11646	11646	11646	11638
Functional Nodes Explored	370	55	206	83	510	46
Security Nodes Explored	1922	1650	152	30	28	19
Beneficial Cuts	40	20	34	23	26	18
Harmful Cuts	453	70	237	81	68	74
Iterations Needed	494	91	272	105	95	93
Time in Model (s)	283	40	319	112	76	273

TABLE 4.1: Experimental results from applying the DOCSDN framework to an order 4 Fat-tree. Each column refers to a separate experiment where the number of paths per source-destination pair given to the framework were varied. Note that the functional objective values in this table are calculated without the cut reward, the second term in Equation 4.8, in order to facilitate comparisons across columns.

observations are in order:

- The overall objective reflects both the functional and security layers. The other objective rows refer to each layer objective individually. The network risk rows quantify the risks and their change as the optimization proceeds. For instance, for the 10 paths benchmark, the risk degrades from 10425 to 11646 or 11.7% as a result of supporting an additional 4 good flows.

- The "nodes explored" rows indicate of the size of the branch and bound tree and remains quite modest throughout.

- Within each experiment we observe a meaningful search, as seen by numerous cuts sent back to the functional layer, to segregate high and low risk flows.

- All runs blocked all flows that contain a high risk host while preserving the low risk flows. All experiments delivered final configurations that preserved the same low-risk flows. We therefore hypothesize that a column-generation would quickly settle down and prove that no additional path can improve the quality

of the solution. It is nonetheless interesting that adding more paths does not negatively impact the overall runtime.

- The objective functions of the functional and security layers use "scores" meant to ease the interplay between the two. Yet, it is wise to consult the *raw* properties of the solutions to appreciate the impact of the optimization. In particular, the number of flows blocked and the network risks. What is readily apparent is that improving functionality induces a slight degradation in the network risks, underlying the conflicting nature of the two objectives. The individual objective scores while moving in the correct direction are not to be viewed as stand alone metrics to determine solution quality but rather inter layer communications indicating improvement or decline from a functional or security perspective.

- The objective scores vary across our experiments due to the stochasticity introduced by our heuristic-driven feedback module (see Section 4.3.4 for discussion). For instance, the functional objecive in the 30 path experiment is slightly worse than it is in other runs, but this difference does not impact the number of serviced flows in the final configuration.

- The variance in time, iterations and number of cuts produced by each experiment is due to symmetries in the formulation. Solutions that are symmetric in the functional layer may not be symmetric in the security layer and induce slightly different solutions there. This is especially true for a Fat-tree network due to its built in redundancy/symmetry.

- Beneficial cuts reflects the number of segregation proposals from the security layers that are adopted by the functional layer (these cuts remove the current

best feasible solution). harmful cuts are segregation proposals that do not "cut" the current best feasible solution or worsen the functional solution.

4.5 Conclusion

Our framework is portable with respect to network risk assessment. Since the risk calculation/analysis is decoupled from the optimization model, the framework can be combined with any procedure that calculates risk on a per path basis. Along with this procedure, the other requirements for implementing a different risk mechanism are 1) A way of evaluating how risk changes due to the deployment of network defenses and 2) The ability to propose candidate cuts that can be passed to the functional layer.

Our results show it is possible to effectively, automatically, and quickly find a network configuration that meets multiple conflicting properties. Our framework is modular, enabling integration of new desired properties. DocSDN will allow network administrators to effectively prioritize and choose their desired properties. The efficiency of DocSDN is enabled by the feedback/interplay between the functional and security optimization layers.

Chapter 5

Functionality and Security - Integrated Framework

5.1 Introduction

Network engineers rely on network appliances to assess the network state (load, good and bad data flows, congestion,...) and public vulnerability databases and security appliances to understand risk. Network engineers have to *integrate* both sources to *assess* the overall risk posture of the network and *decide* what to do.

Due to the intractability of this job, vulnerabilities exist in enterprise networks for long periods: recent work found it can take over 6 months to achieve 90% patching [69] (similar findings in prior studies [107]). Furthermore, some vulnerabilities are publicly disclosed before patches are available or tested, creating a vulnerability window where patching cannot help. The goal of this work is adjust the network in this vulnerability window to mitigate risk and maximize functionality.

(Probabilistic) Attack graphs are used to model risk [106]. An attack graph is a labeled transition system that models an adversary's capabilities within a network and how those can be elevated by transitioning to new states via the exploitation of vulnerabilities (e.g., a weak password, a bug in a software package, the ability to guess a stack address,...). In this work, we focus on risk that is due to network configuration. See, for example, the TREsPASS project for how to incorporate other aspects of risk [76].

Attack graphs can be used to discover paths that an adversary may use to escalate his privileges to compromise a given target (e.g., customer database or an administrator account). Estimations of the probabilities of success of paths coupled with the value of the targets characterize the risk assumed by the network owner [36, 64].

Deciding how to mitigate risk is more delicate. While modern attack graphs can issue recommendations that indicate which edges are most critical [10, 36, 54, 55, 96, 98, 105] (which exploits should be patched, where a firewall must be stood up, etc...) to decrease the overall risk in the network, they do not account for the loss in functionality (i.e., the collateral damage) that they induce. Furthermore, recommendations must be implemented manually increasing response time. A more desirable scenario to cope with emerging threats is:

1. A security appliance identifies a problematic flow/user (signaling a change in a component's risk) or a new vulnerability is published in CVE (Common Vulnerabilities and Exposures) [32],
2. An attack graph is generated,
3. Recommendations are derived from the resulting graph, and
4. Recommendations are programmatically deployed and implemented

The challenge to deliver this vision is threefold: first, to evaluate attack graphs quickly; second, to incorporate functionality requirements and; third, to quickly and transparently deploy recommended changes.

Our contribution Our contribution is an optimization framework we call FASHION (Functional and Attack graph Secured HybrId Optimization of virtualized Networks). FASHION considers both functionality and security when deciding how to configure the network. The *functional* layer treats network traffic as a multi-commodity data flow problem and provides the logic to route flows. It ensures that any routing solution carries each flow from its source to its destination (if the flow is not dropped due to a security layer decision), respects link capacities and network device throughputs, and satisfies the required demand. To enable the security layer, we introduce security metric which can be evaluated using linear programming to deliver quick calculation of risk on related networks. The *security* layer then integrates the risk of a configuration to create a joint model between the two layers. This joint model (solved with integer linear programming) focuses on reconfiguring the network. All FASHION code is available at [23].

The risk measure A major problem in attack graphs is their scalability [9,94] as they consider all paths an attacker could take to achieve their objective. Two common graph representations are an attack *dependency* graph and an attack *state* graph. In the *dependency* view each node represents an exploit or a capability in the network. The main drawback of the dependency representation is that analysis of overall risk is difficult. The second representation is the attack *state* graph. In this representation

each node represents an attacker's current capabilities. This representation simplifies analysis; however, an exponential blowup in representation size makes it prohibitively expensive for moderate size networks [50, 94].

Several works have used optimization to create attack graph recommendations [36, 67]. To the best of our knowledge, there has been no analysis of attack *dependency* graphs which is conducive to repeated evaluation on related graphs (differing by the introduction of a defensive countermeasure or a new flow). FASHION must quickly consider many functionality and security considerations, meaning that state of the art recommendation engines are too slow. Instead, we develop an *approximation* of a prior measure described by Wang et al. [121] (described in Sec. 5.2) and consists of a weighted sum of two parts:

Reach: The total impact of the nodes that are reachable by the attacker. This translates to an attack graph where each nonzero probability edge is assumed to be compromised. The first generation of attack graphs considered this measure [56, 60, 94, 112].

Path: The risk (impact times likelihood) of the maximum path in the network. Prior work by Khouzani et al. [64] used this measure in attack *state* graphs (see Section 5.1).

Since this is an approximation, one may wish to validate the output of fashion using the full risk measure or existing network security or functionality appliances (see Figure 5.1). If a validation tool does not approve the configuration, it may be possible to programmatically incorporate its output. If the tool outputs an important flow that is not being served, the value of this flow can be increased in the configuration

(see an example in Table 5.1). If the tool outputs a flow that *should not* be served, the risk of the endpoint can be increased in the configuration.

To understand the quality of our metric when evaluating FASHION (Section 5.4), we use Wang et al.'s algorithm as ground truth for the evaluating the security of our resulting configurations. In all generated instances our metric of $avg(\mathsf{Reach}, \mathsf{Path})$ is monotonic in Wang et al's algorithm (which is far too slow to be used in an optimization). FASHION usually outputs a configuration in under 30 minutes, allowing response to short term events (on networks with 432 hosts and 181 networking devices). Our approach can incorporate the idea of equivalence classes used in previous attack graph research to further improve scaling [55].

Integrating Fashion To address the deployment challenges, we focus on software defined networks (SDN) and recommendations that can be implemented through control APIs of SDN controllers. For concreteness, we evaluate our approach on data-center networks which frequently use virtualized networking [5]. FASHION is not applicable for all networks at all times. FASHION is not an "add on" to the existing infrastructure of a legacy system.

FASHION's output is fed to a tool which creates a high-level SDN controller implementing the configuration. Specifically, FASHION's output interfaces with a Frenetic controller [44]. Figure 5.1 shows how FASHION fits into a network deployment pipeline. It can be rerun anytime there is a change in the environmental setup.

We assume *white list* routing where only desirable flows are carried to their destination. This corresponds to all extraneous flows in the network already not being served. This places FASHION in the region where there is a sharp tradeoff between

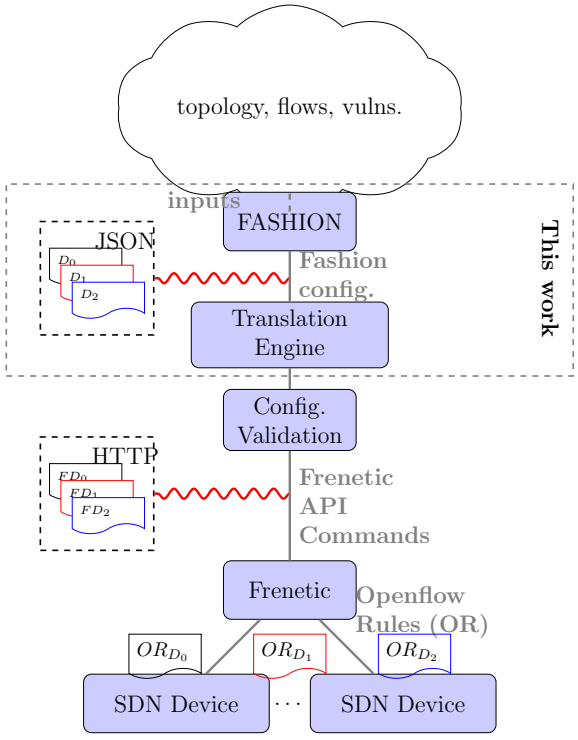
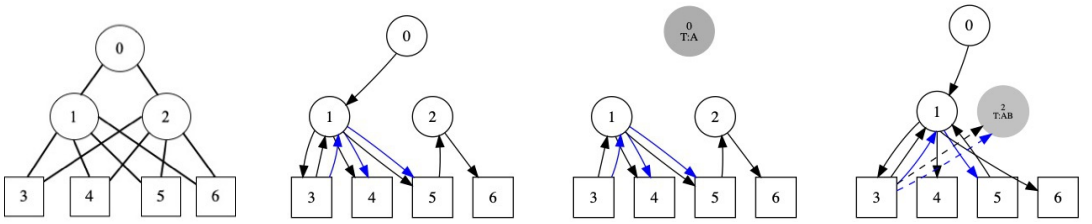


FIGURE 5.1: FASHION takes as input the existing physical topology, required flows (including bandwidth and value), and vulnerabilities. The FASHION optimizer outputs a JSON file describing how to route flows in the network (and flows that should be blocked). This JSON is fed to our Python Frenetic Translation Engine which interacts with the Frenetic controller to create the necessary Openflow tables in the SDN devices.

functionality and security. That is, any reduction in risk corresponds to a reduction in functionality. If one has an attack graph that routes using flows that are not necessary to functionality, this routing can be disabled at no cost to functionality. Furthermore, we assume devices can use *source routing* which allows more flexibility in our security decisions and ability to respect capacity. We consider routing between individual hosts. However, FASHION is applicable for optimizing functionality and security between subnets.



(A) Physical network topo. (B) Functional config. (C) Security config. (D) Balanced Config.

FIGURE 5.2: Output of optimization model on sample network when considering functionality only, security only, and both. The set of required flows and existing exploits are shown in Table 5.1 and the corresponding attack graph in Figure 5.3. Figure 5.2a shows the set of available network links that can be used by the optimization framework. Figure 5.2b shows the output of the model when considering only functionality requirements, Figure 5.2c shows the output when considering only security requirements, and Figure 5.2d shows a solution respecting both requirements.

Driving Example We now describe a toy example that is used to illustrate FASHION's key concepts. Recall that the objective of the framework is to produce a collection of decisions to configure the network devices (routers, firewalls, ...) to serve the functional requirements while minimizing the risks incurred by the network. In this work we consider routing decisions on flows only (including blocking a flow).

Figure 5.2a illustrates the physical layer of the network.[1] It features 3 SDN appliances, nodes 0, 1 and 2 that route traffic as well as block it (act as firewalls). The toy network features 4 hosts, 3 through 6. This mini network can support different paths between the SDN appliance 0 and any host (e.g., host 3).

This physical network must be configured to serve traffic demands. Table 5.1 shows a collection of data flows in the form $s \rightarrow t$ conveying that traffic emanating from node s must reach node t. Each data flow has a type (here A or B). Each data

[1]The following conventions are used: circle are routers, square are hosts, and black lines are physical connections.

src → dst	type	value
0 → 3	A	$$$
3 → 4	A	$
3 → 4	B	$
3 → 5	A	$$
3 → 5	B	$
5 → 6	A	$$

Preconditions	post
(3,0: A)	(3,1)
(5,0: A)	(5,1)
(6,0: A), (5,1)	(6,1)
(4,0: B), (3,1)	(4,1)

TABLE 5.1: Flows (left) and exploits (right) in the network in Figure 5.2.

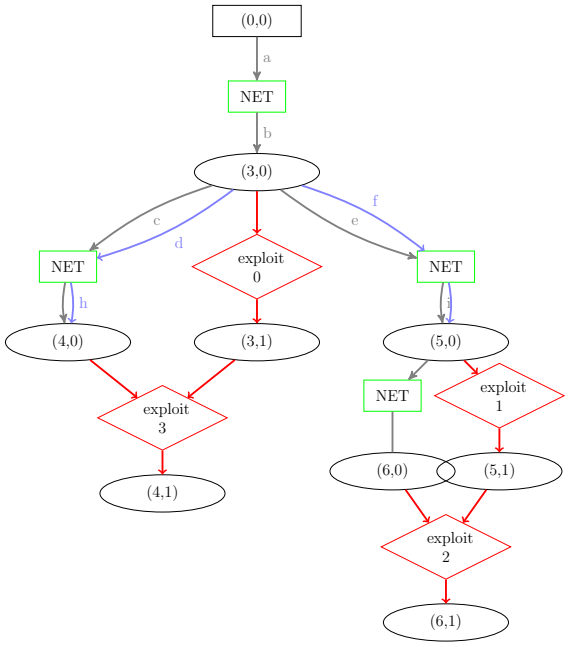


FIGURE 5.3: Attack graph for network in Figure 5.2

flow also carries an economic value shown by the number of $ signs. Table 5.1 also shows exploits, each exploit has one or more pre-conditions to be triggered as well as effects. A pair (h, p) states that the adversary secured capability p on host h where the ordered set of capabilities is $\{0, 1\}$. Privilege level 0 represents the ability to send traffic to that host, it is augmented with the traffic type (either A or B). For the toy example, we assume all exploits have probability 1 of being achieved if preconditions are met.

Figures 5.2b-5.2d show possible outputs of the framework, i.e., a configuration that defines routing tables for each SDN appliance (including explicit firewalls). In this network configuration, the following conventions are added: gray nodes block traffic, black (resp. blue) arrows are type A flows (resp. B), and dashed arrows represent blocked traffic.

Figure 5.3 conveys the attack graph for this network.[2] For instance, from having capability 0 on host 0 (the entry point) one can transition to having capability 0 on host 3 (host 3 is reachable from SDN device 0). The red diamond node exploit 0 shows that exploiting the vulnerability on host 3 will deliver a privilege escalation, i.e., capability 1 on host 3. Figure 5.2b shows a configuration resulting from FASHION in which the objective was exclusively the maximization of the functional objective (total flow). All data flows are served and no counter-measures are deployed, leaving the network exposed to an adversary as shown by the existences of paths that reach the target nodes $(6, 1)$ and $(4, 1)$ (in Fig. 5.3).

Alternatively, Figure 5.2c conveys a configuration at the opposite end of the spectrum where security is paramount. The attack graph in Figure 5.3 shows that the easiest way to block access to the 4 exploit nodes is to sever the edge b $(0 \rightarrow 3)$. Indeed, if one cannot reach host 3, exploit 0 is not usable and the hosts 4 and 5 are unreachable preventing the attacker from leveraging exploits 1 through 3. In Figure 5.2c node 0 is now blocking all traffic of type "A." In this configuration, the internal traffic proceeds unabated. However, the high "commercial" value of flow $0 \rightarrow 3$ was not respected doing great damage to the economic value of the network

[2]Figure 5.3 uses the following conventions: 1) the square node is the entry point of the network, 2) green square NET nodes represent network reachability, 3) diamond nodes are exploits, 4) circle nodes are (host,privilege) states, 5) black (resp. blue) arrows correspond to network connection of type A (resp. B), 6) incoming red links are precondition states of exploits, and 7) outgoing red links are postcondition states of exploits.

service.

Finally, in Figure 5.2d the interplay between competing priorities become apparent. In this solution, the flow $0 \rightarrow 3$ is served because of its intrinsic economic value while edges d and e in the attack graph (corresponding to routing $3 \rightarrow 4 : B$ and $3 \rightarrow 5 : A$ respectively) are turned off. More precisely, these flows are routed to node 2 which is a firewall for both traffic types as shown in Figure 5.2d. This is done to prevent the attacker traversing edges h and i and then being able to use exploits $1, 2, 3$. This unconventional placement of countermeasures prevents an adversary from reaching the capability nodes $(6, 1)$ and $(4, 1)$ while still preserving the most valuable flow.

Further related work Attack graphs are not a panacea. Major criticisms of attack graphs include the difficulty finding the necessary inputs [4, 97], the difficulty in implementing the output recommendations [78], and scaling issues. FASHION's risk model is designed for fast evaluation on related networks. As necessary, one can include more expensive risk metrics in Configuration Validation before deploying the resulting configuration. Indeed, to implement the attack graph recommendation, it is necessary to *interpret* its output, a task deemed too difficult and too remote from the network to be actionable [19]. FASHION is designed to address this problem.

Scalability remains a limitation (further discussion in Section 5.5). The size of attack graphs is a weakness, driving the time it takes to generate and analyze them. The evolution of networking environments towards SDN provides the perfect opportunity for integration with attack graphs. SDN offers a centralized control and holistic view of the network no longer requiring external scanning tools to discover reachability

data [39].

Researchers have proposed high-level SDN programming languages in order to efficiently express packet-forwarding policies and ensure correctness when dealing with overlapping rules [44, 101]. These languages focus on parallel and sequential composition of policies to ensure modularity while providing correctness guarantees. Importantly, when our framework proposes a set of new rules for the controller it is necessary to ensure that good traffic is not lost [102].

To aid in network configuration, research tools assess network reachability [65], network security risk [106, 126], and link contention [114, 122]. These tools assess the quality of a configuration with respect to a single property and do not provide recommendations.

Recent tools generate network configurations from a set of functional and security requirements [14, 38, 81, 89, 117]. Captured security requirements include IPSec tunneling, allowing only negotiated packet flows, and ensuring identical rule sets on all firewalls.

Our work can be seen as unifying two recent works, one by Curry et al. [33] and another by Khouzani et al. [64]. Curry et al. proposed an optimization framework for deciding on a network configuration based on the given desired network functionality of data flows and the underlying physical network. Curry et al. showed how to produce a network configuration that meets all demands while blocking adversarial traffic. In their model, each network node has an input risk and nodes assume a fraction of the risk of any node with which they share a path. However, their risk measure does not take into account the ability of an adversary to pivot in the network.

Khouzani et al. [64] created an optimization engine designed to minimize security risk as represented by an attack graph. They show how to formulate the *most effective*

attack path of an attack graph using a linear program. Their functionality view is limited to imparting an explicit numeric functional *cost* to each remedial action. It is unclear how to create these input costs used in the model or what to do if the functional requirements are non-linear. Importantly, Khouzani et al.'s formulation requires *attack state graph*. In practice, attack state graphs are exponential in the size of the network, as each set of capabilities is a distinct node [50]. Our Path measure also captures the maximum likelihood path. However, since our measure operates on an attack *dependency* graph it corresponds to a different property. We discuss this in the Section 5.2.

Organization The organization of this work proceeds as follows. Section 5.2 introduces background on attack graphs and on the measures we will optimize over, Section 5.3 documents the optimization model, Section 5.4 evaluates FASHION, and Section 5.5 concludes.

5.2 Attack Graphs

FASHION's goal is to balance the functionality and security needs of the network. Functionality needs are relatively straightforward to state: a set of desired network flows that should be carried in the network while respecting link capacity. Security is more complicated to state. We use the abstraction of *attack graphs*.

Attack graphs model the most likely paths that an attacker could use to penetrate a network [9, 56, 106, 112]. In the attack graph model attackers traverse the network to reach their target(s) network resource. This traversal may combine traditional

network capability such as routing with exploitation of a software/hardware vulnerability. An attack graph assumes an attacker starts at some entry point such as a publicly facing Web page and through a series of privilege escalations and network device accesses pivots to eventually reach his or her desired destination. (The technology supports an arbitrary starting point if one wishes to consider insider attacks.) We focus our discussion here on attack *dependency* graphs as defined by Homer et al. [51]. Another common type of graph is called an attack *state* graph. We discuss the distinction between these two data structures in Section 5.2.1. For brevity, we just refer to attack dependency graphs as attack graphs.

There are two types of nodes in an attack graph: capabilities, denoted C, and exploits denoted as ex. An exploit requires some number of capabilities as preconditions. If an attacker has successfully obtained those preconditions they are assumed to gain all successors of ex with some probability. Example capabilities include ability to send packets, user level authority, root level authority. Example exploits include an SQL injection which requires an ability to send packets and a SQL server running a specific software version. A successor of this exploit node may be root level access on the device running the SQL server. In the attack graph in Figure 5.3, when an attacker executes Exploit 0, they achieve privilege level 1 on node 3 with some probability.

Building an attack graph requires network reachability information, device software configurations and known exploit information [32, 41, 86, 92] to generate the graph. Attack graphs are only effective in measuring how an attacker would traverse using known vulnerabilities and system state. While it is possible to consider the implications of a new vulnerability [55] this is not a native capability.

In isolation a misconfigured device which allows unauthorized access may be be-

nign but when coupled with network access to ex-filtrate data or pivot to additional targets the results can be devastating. The goal of constructing and analyzing an attack graph is to understand the security posture in total. An attack graph should allow one to understand defensive weaknesses and critical vulnerabilities in the network. Since all enterprises have limited budgets, the goal of this analysis is usually to prioritize changes that have the largest impact.

5.2.1 The size of attack graphs

Since their introduction, a major problem in attack graphs is their scalability [9, 94]. They consider all paths an attacker could take to achieve their objective. There are two very different ways of representing the graph that are called an attack *dependency* graph and an attack *state* graph. In the *dependency* view each node represents an exploit or capability in the network. It may be possible to achieve a capability using many different paths. Furthermore, multiple conditions may be necessary to achieve this capability, for example, network reachability of a database machine and a SQL injection attack. This is the view presented in Figure 5.3.

In the most general form, each exploit has an associated Boolean structure (indicating when the exploit can be obtained) and a probability (indicating attacker success rate in carrying out the exploit). Capability nodes are annotated with an impact value that signifies the cost of an adversary achieving that capability (following the NIST cybersecurity framework guidance [91]). In this work, we focus on exploits that are AND and OR prerequisites (and not arbitrary Boolean formulae). Note the dependency representation may have cycles. The main drawback of the dependency representation is that analysis of overall risk is difficult. Even if one assumes that

probability associated with each edge is independent, calculating the overall probability requires consideration of all paths, and there may be infinite paths from the starting point to a target if the graph has cycles. We return to this problem after describing the state representation and prior work on quickly evaluating attack graphs on similar enterprises.

The second representation is the attack *state* graph. In this representation each node represents an attacker's current capabilities. Suppose there are k capabilities in the network. In the state representation, there are 2^k nodes representing whether the attack currently has each of the capabilities $1, ..., k$. In the dependency representation, the attack graph has k capability nodes and some number of exploit nodes. This representation is acyclic and makes it very easy to carry out analysis [50]. However, an exponential blowup in representation size makes it prohibitively expensive for moderate size graphs [50]. Khouzani et al's model [64] requires the state representation to minimize risk.

5.2.2 Evaluating related attack graphs

In this work we focus on the ability to repeatedly evaluate an attack graph on related networks. The ability to perform this analysis quickly is critical to utilizing attack graphs in our optimization framework. The ability to regenerate the attack graph multiple times at decreased cost has been addressed recently in related contexts.

Almohri *et al.* [8] considered an attack graph setting where the defender has incomplete knowledge of the network. An example source of this uncertainty is mobile device movement. They then construct attack graphs using a probabilistic model which includes the uncertainties in network configuration. In our setting, we are try-

ing to find the best configuration under a variety of settings, Almohri *el al.* would be appropriate if the functionality and security requirements could not be unified as it could provide recommendations under a variety of related functional settings.

Frigault and Wang [46] argue that it is inaccurate to measure probabilities with a fixed probability of exploit. They argue that factors such as patches being available will decrease the threat while wide spread distribution of vulnerability details may increase the threat. As such, they conduct attack graph analysis where the graph is static but probabilities can change over time. Poolsappasit et al. [98] also argue that the probability of attack success changes over time.

Note that if one is willing to consider a *complete* attack graph then all changes in the graph can be represented with a change in probability. However, this is akin to considering the state representation as one needs to consider an edge from each subset of nodes.

To the best of our knowledge, Khouzani et al.'s work is the only work that considers defensive actions that can drastically change the attack graph and a formulation of the risk calculation graph that is amenable to optimization. However, their model is inherently tied to the state graph representation.

5.2.3 Formalizing the problem

This section defines the risk measure used as ground truth in FASHION. The metric is drawn from Wang et al. [121] augmented with impact for each node. Wang et al.'s metric assumes an attack graph where the probability of achieving each exploit is independent. Since paths in an attack graph often overlap, the probability of achieving prerequisites of an exploit may not be independent. This simplifying assumption is

often used since considering correlated probabilities makes the problem significantly harder [51]. Consider the following notation:

- Let $\mathsf{EX} = \mathsf{EX_n} \cup \mathsf{EX_v}$ be the set of all exploits with $\mathsf{EX_n}$ the network reachability exploits (to model a configurable network) and $\mathsf{EX_n}$ be the set of vulnerabilities.
  - Further assume that $\mathsf{EX} = \mathsf{EX_{AND}} \cup \mathsf{EX_{OR}}$, i.e., the set of exploits can be partitioned into conjunctive or disjunctive nodes.
  - Assume that $\mathsf{EX_n} \subseteq \mathsf{EX_{OR}}$, i.e., reachability is treated as OR between multiple traffic types among hosts.
  - Exploits $\mathsf{ex} \in \mathsf{EX}$, carry a probability $p(\mathsf{ex})$ of exploitability given that all prerequisites have been satisfied. It can be estimated using vulnerability databases [32, 41, 92]. Note that $p(\mathsf{ex})$ is a component metric, we seek to capture the cumulative risk in the network (see discussion in [113, 121]).
- Let C be the set of all capabilities in the network. A capability C carries an impact $\mathsf{Pact}(\mathsf{C}) \in \mathbb{R}^+ \cup \{0\}$.
- Let $N = \mathsf{EX} \cup \mathsf{C}$ denote a node set.
- Let $E = R_r \cup R_i$ denote the edge set with
  - $R_r \subseteq \mathsf{C} \times \mathsf{EX}$: the exploit's prerequisites.
  - $R_i \subseteq \mathsf{EX} \times \mathsf{C}$: capabilities gained from an exploit.

  There are no edges in $\mathsf{C} \times \mathsf{C}$ or $\mathsf{EX} \times \mathsf{EX}$.
- An attack graph $G(N, E)$ is a directed graph.
- Let $\mathsf{Start} \subseteq \mathsf{C}$ denote a set of capabilities that the adversary is believed to have.

For any node n, $\mathsf{Pred}(n) = \{v|(v,n) \in E\}$ denotes all its predecessors in G, while $\mathsf{Succ}(n) = \{v|(n,v) \in E\}$ denotes all of its successors.

Basic Risk First, consider how to compute the overall risk in the absence of cycles. This discussion heavily follows Wang et al.'s [121] methodology. Their formulation is centered on Bayesian inference and is augmented with an impact for each capability node.

The primary goal is to compute cumulative scores $P(\mathsf{ex})$ and $P(c)$ for each node in G. These represent the likelihood that an attacker reaches the specified node in the graph.[3] From these scores one may consider the overall risk:

$$\mathsf{Risk}(G) = \sum_{c \in C} P(c) * \mathsf{Pact}(c) \tag{5.1}$$

To compute $P(\cdot)$ first consider the acyclic case. For AND nodes, the cumulative score is the product of all predecessors and $p(\mathsf{ex})$. For OR nodes, the cumulative score is the sum of all predecessors component score minus the product of each pair of probability scores (using Bayesian reasoning). All capability nodes are treated as OR nodes. For a set X, we define the operator $\mathsf{Bayes} : X \to [0,1]$ as:

$$\mathsf{Bayes}(X) = \begin{cases} \sum_{X' \subseteq X} (-1)^{|X'|+1} \prod_{x \in X'} p(x) & X \neq \emptyset \\ 0 & X = \emptyset \end{cases}.$$

Definition 5.2.1 (Network Risk)**.** *[121, Definition 2] Given an acyclic attack graph G and any component score assignment function $p : \mathsf{EX} \longrightarrow [0,1]$, the cumulative*

[3]It is possible to assign individual component scores $p(c)$ for nodes $c \in \mathsf{C}$. In this work we assume that $p(c) = 1 \; \forall \, c \in \mathsf{C}$. Without loss of generality, we assume that all uncertainty in the attacker's success is represented in exploit edges.

score function $P : \mathsf{EX} \cup \mathsf{C} \longrightarrow [0,1]$ *is defined as*

$$P(\mathsf{ex}) = \begin{cases} p(\mathsf{ex}) \cdot \prod_{c \in \mathsf{Pred}(\mathsf{ex})} P(c) & \mathsf{ex} \in \mathsf{EX}_{\mathsf{AND}} \\ p(\mathsf{ex}) \cdot \mathsf{Bayes}(\mathsf{Pred}(\mathsf{ex})) & \mathsf{ex} \in \mathsf{EX}_{\mathsf{OR}} \end{cases}$$

$$P(c) = \begin{cases} 1 & c \in \mathsf{Start} \\ \mathsf{Bayes}(\mathsf{Pred}(c)) & \textit{otherwise.} \end{cases}$$

Note that $P(n)$ can be computed for all $n \in N$ as long G is acyclic. $P(n)$ can be computed once $P(e)$ is known for all $e \in \mathsf{Pred}(n)$. Thus, there exists at least one topological ordering that allows this computation (and all topological orderings result in the same computation). For a given acyclic attack graph let algorithm $\mathsf{ARisk}(G)$ compute $P(n_i)$ for $n_i \in N$.

Handling Cycles The adaptation of Wang et al's [121] to handles cycles is used to evaluate security measures introduced in subsection 5.2.4. Wang et al. [121] observed that:

1. Cycles with no entry point can be safely ignored and all nodes can be set to 0 likelihood.

2. We only need to measure the probability for the first visit of a node. Consider a cycle with only one incoming edge. Denote by n the node with the incoming edge. We can safely compute $P(n)$ and ignore the cycle as any path that traverses the cycle will have already included n. Thus, the edge closing the cycle at n can be ignored.

3. Cycles with multiple entry points are more delicate to handle and require the

removal of an edge from the graph. The key insight is no path that an attacker traverses will actually follow the cycle. Different paths will include subsets of edges from the cycle.

Wang et al. [121] propose the following methodology for handling cycles with multiple entry points. Assume that all nodes that can be topologically sorted have been and their cumulative probabilities are assigned. Let X be a cycle with at least two entry points. For each entry point $x \in X$ one can compute $P(x)$ without considering $\mathsf{Succ}(x)$. While x's successors are important in calculating the overall risk they do not impact $P(x)$. Compute a new attack graph G' which has all $\mathsf{Succ}(x)$ removed and use it to calculate $P(x)$. Importantly, the graph G' may still have cycles which inhibit computation of $P(x)$ but this process can be done inductively. Once it terminates it is repeated for all entry points in the cycle. Once all entry points have their likelihood, the rest of the cycle can be safely evaluated and ARisk continues.

Some parts of the computation can be reused throughout the recursion (all nodes that were sorted before the cycle was encountered) but the likelihoods computed in this process for $x' \neq x$ are not the true likelihoods and cannot be reused between recursive steps.

When the paper uses the term $\mathsf{Risk}(G)$ it refers to this metric and calculation. As discussed in Section 5.4 the algorithm is too slow to be incorporated into an optimization framework that is considering many possible solutions. The Python implementation has been open-sourced along with the rest of FASHION [23].

5.2.4 Approximating Risk

In order to incorporate cumulative risk into an optimization framework, we turn to approximations of risk that can be linearized. The risk calculation presented in Definition 5.2.1 and its augmentation for handling cycles is non-linear. To the best of our knowledge, even a closed form of the calculated value is not known. We consider two approximations to serve as proxies. After introducing these approximations, we remark on the strengths and weaknesses of both of these approximations and why they complement each other well. We defer to Section 5.4 to remark on quality of our measures.

We call these two approximations *Reachability* and *Max attack*. The qualitative differences in the two approximations may make either measure (or a combination) appropriate for a given set of network parameters (scale of network, traffic, pattern, vulnerabilities).

Reachability Instead of using the raw probability of exploitation, we binarize them.

$$P^*(n) = \begin{cases} 1 & P(n) > \theta \\ 0 & P(n) \leq \theta \end{cases}$$

The value θ is a threshold used to determine whether a capability should either be ignored or considered attained. We consider $\theta = 0$. Since $p(n) \in \{0, 1\}$, we can apply standard Boolean linearization techniques to get a tractable representation of dynamic risk.

$$\mathsf{Reach}(G^*) \stackrel{def}{=} \sum_{c \in C} P^*(c) * \mathsf{Pact}(c) \tag{5.2}$$

Utilizing the binary representation of exploits in Equation 5.2 is an approximation, measuring how an attacker can impact a target network. Since we consider $\theta = 0$ this measures the total impact of nodes reachable by the adversary. This is equivalent to calculating the weighted size of the connected components in G that contains Start. The first generation of attack graphs did not consider annotate nodes with probability and measured this quantity [103, 112]. This models the *worst case* approach when calculating attacker compromise of network capabilities and assumes that the all vulnerabilities with probability of exploitation above a threshold will be successfully exploited.

However, this approach does have a weakness when the goal is to jointly optimize functionality and security. Consider two nodes a and b where $\mathsf{Pact}(a) = 2 * \mathsf{Pact}(b)$. Further suppose, at least a or b must remain in the connected component to satisfy functionality demands. The reachability metric will prioritize disconnecting b. However, it may be that the attacker is less likely to reach b and this decision is not optimum. This may be the case even if $p(b) \geq p(a)$ (it is possible that $P(a) > P(b)$ even in this case due to the likelihood of reaching their predecessors).

Max Attack The second risk measurement we introduce is the attacker's most likely course of action. This measure is based on Khouzani et al.'s *most effective attack measure* [64]. We introduce the measure first and then say how our approach differs from Khouzani et al.

Let σ be the attacker's starting point in the attack graph. We define $\omega_{\sigma \to c}$ to be the set of all paths, where a path is sequence of edges $(e_1, \ldots, e_k)$ such that $e_i \in E$, from σ to c in the attack graph. Let $\Lambda_c \in [0, 1]$ be the normalized impact of an attacker obtaining capability c. Then the most effective attack path is defined as follows.

$$\mathsf{Path}(G) = \max_{c \in \mathsf{C}} \Lambda_c \max_{\omega_{\sigma \to c}} \prod_{e \in \omega_{\sigma \to c}} p(\mathsf{Pred}(e)) \tag{5.3}$$

Instead of having multiple targets, an auxiliary target μ is considered, that will be the sole target capability of the attacker. To do this, edges from each $c \in \mathsf{C}$ to an auxiliary exploit ex_c are introduced, such that $p(\mathsf{ex}_c) = \Lambda_c$. Then we introduce an edge from each ex_c to μ. Doing this, we can reformulate equation 5.3 as:

$$\mathsf{Path}(G) = \max_{\omega_{\sigma \to \mu}} \prod_{e \in \omega_{\sigma \to \mu}} p(\mathsf{Pred}(e)) \tag{5.4}$$

However network defenses can be deployed in order to reduce the probabilities of these exploits. Let $x_d \in \{0, 1\}$ be a binary decision variable denoting whether a network defense d has been deployed. Let $p_d(\mathsf{ex})$ be the reduced probability of exploit ex due to network defense d. With this, the probability of exploit ex with respect to network defense decision x_d is given by

$$p_{x_d}(\mathsf{ex}) = p(\mathsf{ex})(1 - x_d) + p_d(\mathsf{ex})x_d. \tag{5.5}$$

Therefore we want to minimize the risk due to the most effective path risk over all the possible configurations of network defenses available. This approach identifies appropriate locations to deploy network defenses in order to protect both high value capabilities with low exploitablity as well as lower value, more exploitable assets. We will incorporate these defense decisions when defining our optimization model in section 5.3.

Note that Path does not distinguish between a graph with 2 paths with the same underlying probability and a single path with that probability. In addition to this

inaccuracy (that was present in Khouzani et al.'s work) working on the attack dependency graph introduces two sources of error:

1. Path only counts the impact from the last node on the path. This is because it is only measuring the probability of a path and the impact is added as a "last layer" in the graph. So it doesn't distinguish between two paths (of equal probability) where one path has intermediate nodes with meaningful impact. In the attack state representation each node has the current capabilities of the attack and thus impact of this node set can be added as a last layer.

2. The path used to determine Path may not be exploitable by an adversary due to AND nodes. That is, the path may include a node with multiple prerequisites and the measure only computes the probability of exploiting a single prerequisite. In the attack state representation there are no AND nodes so this problem does not arise.

Balancing Reach and Path. As described above both Reach and Path have inherent weaknesses. We believe (and demonstrate in Section 5.4) that FASHION outputs better solutions when considering both metrics. We call the weighted sum of these two functions Hybrid. This is because Reach and Path mitigate each other's weaknesses. However, even a joint optimization over both metrics is still heuristic.

Reach is effective at isolating nodes that don't need to communicate. However, when there is a tie between two nodes that could be excluded, Reach may not make the right decision because it does not know the likelihood of reaching these nodes. However, Path's measure can effectively break ties on which node to remove based on the maximum likelihood path that reaches the node.

Path is effective at isolating nodes that have a high probability path to them.

Yet, this measure does not account for the other nodes compromised "on the way" to the target node. By minimizing the total weighted reachable set using Reach this weakness is partially mitigated. Similarly, if two paths have similar probability but one contains multiple AND prerequisites, it may have a larger reachable component, enabling Reach to again break ties.

5.3 Optimization Model

This section highlights the content and structure of the optimization model used to obtain network configurations that uphold a balance between functionality and security.

The optimization model is a *binary integer programming* (BIP) model as all decision variables are binary. The model contains two components dedicated, respectively, to the modeling of the data network and its job as a carrier for data flows and to the representation of the attack graph and the modeling of induced risk measures Reach and Path. The core decision variables fall in two categories.

First, Boolean variables modeling the routing decisions of the data flows in the network are associated to network links and subjected to flow balance equations as well as link capacity constraints that capture the valid delivery of flows and the functional rewards associated to those deliveries based on the flow values.

Second, Boolean variables are associated to the deployment of counter-measures in the network. In this work, counter-measures are routing a flow to a firewall (rather than its destination). Auxiliary Boolean variables are introduced to facilitate the expression of the model and setup *channeling constraints* that tie the attack graph

model to the network routing model so that routing as well as blocking decisions are conveyed to the attack graph side and result in severing arcs that express pre-conditions of exploits.

While the constraints devoted to capturing reachability, i.e., Reach, are relatively straightforward, the modeling of the most effective path, i.e., Path is more delicate for two reasons. First, it requires the use of products of probabilities held in variables yielding a non-linear formulation. Thankfully, that challenge can be side-stepped by converting those products into sums with a logarithmic transform. Second, it delivers a $\min - \max$ problem that needs to be dualized to recover a conventional minimization.

The objective function combines (linearly) two linear expressions capturing the functional objective and the risk objective. Both are weighted with a parameter α that allows the model user to play with the risk-functionality trade-off. Namely, the objective is of the form $\min \alpha \cdot \mathcal{O}_f + (1-\alpha) \cdot \mathcal{O}_s$ where $\mathcal{O}_f$ is the functional objective and $\mathcal{O}_s$ is the risk objective.

The risk objective $\mathcal{O}_s$ is itself a linear combination of three components:

$\mathcal{O}_d$ The cost of the security measures

$\beta_1 \cdot \mathcal{O}_r$ The weighted Reach risk

$\beta_2 \cdot \mathcal{O}_p$ The weighted Path risk.

Notably, using $\beta_1 = 0, \beta_2 = 1$ delivers a model that exclusively consider Path as its risk measure while $\beta_1 = 1, \beta_2 = 0$ focuses exclusively on the Reach risk measure. Without loss of generality, let $\beta = \beta_1$ and $\beta_2 = 1 - \beta$ with $\beta \in [0, 1]$. The parameterization gives a hybrid risk measure based on a convex combination of Path and Reach. Two

key parameters of the optimization models are, therefore, α and β that control the functionality-risk tradeoff and the balance between Path and Reach measures.

5.4 Evaluation

In this section we show the efficacy and efficiency of FASHION. The goal is to understand (i) does FASHION produce configurations that effectively balance functionality and security? and (ii) does FASHION produce configurations quickly, and what time scale of events can FASHION respond to?

5.4.1 Experimental Setup

The evaluation is made on a Linux machine with an Intel Xeon CPU E5-2620 2.00 GHz and 64GB of RAM.

Network Topology The network topology is as described in chapter 1 discussion of Fat-tree. The benchmarks include small to medium sized instances. The largest instance tested, includes 613 devices (hosts and SDN devices) and 1332 links between these.

Network Traffic Network traffic is as described in Chapter 1 with specific considerations for this chapter as follows. For considered instances, the number of traffic type varies from 1 to 3. One-third of the instances have 1 traffic type, one-third of the instances have 2 traffic types and the one-third of instances have three traffic types. The number of flows per host is varied across benchmarks with steps $\{1, 3, 5, 10\}$ to vary network utilization [35, 61].

Vulnerabilities Synthetic vulnerabilities are injected on hosts within the network.

The generation adopts several components from the vulnerability model presented the recent CVSS 3.1 Base metrics focusing on exploitability and impact [41]. The percentage of *exploitable hosts* ranges in $\{10\%, 20\%, 30\%, 40\%, 50\%\}$ and the average number of vulnerabilities per host (1-5) drives the total number of vulnerabilities injected. The number of vulnerabilities per host is representative of Zhang *et al*'s findings [130] from scans of publicly available VMs after patching was performed.

Each vulnerability has one or more prerequisite conditions and a single post condition of privilege escalation (if successfully exploited). Each vulnerability is uniformly assigned a score $[0 - 1]$ representing the probability of exploitation. Three privilege levels $\{0, 1, 2\}$ are assumed with 0 denoting networking reachability.

Each generated exploit has a single precondition with probability .50, two preconditions with probability .25, and three preconditions with probability .25. Procedurally, the single precondition exploits are generated at random from selected exploitable hosts, each allowing the escalation of a single step in privilege level. If the required number of single precondition exploits exceeds the number of exploitable hosts additional hosts are infected. The multi-precondition exploits are generated by selecting one prerequisite randomly from the pool of existing (single precondition) exploits and generating a new exploit which increases the privilege of one of the input exploits, the other prerequisites are selected randomly from the current set of achievable capabilities.

The impact of a successful exploit is reflective of the value of the threatened host. We uniformly assigned each host an integer in $[1, 100]$ to represent its value to an organization. The impact of compromise of reaching a privilege level on a host is a percentage of this asset value. The quantities $[20\%, 40\%, 100\%]$ are used as the scaling factors for the three privilege levels $\{0, 1, 2\}$.

5.4.2 Results

The discussion focuses on answering the two questions introduced in the beginning of Section 5.4: does FASHION produce good configurations and does it do so in a timely manner? Answering the first question is slightly delicate because our security optimization is using Reach and Path instead of using Risk. Throughout this section we only report on the security quality of the final configuration with respect to Risk. The algorithm for computing Risk is too slow to use in the optimization model but allows an effective check on the quality of the solution a posteriori. In all results the functionality score is the normalized value of the delivered traffic: it considers the total value of traffic delivered when $\alpha = 1$ (corresponding to the optimization considering only functionality) as a functionality score of 1 with the other functionality scores normalized accordingly. Risk is normalized in a similar way and computes the Risk value when $\alpha = 1$ (no protections deployed) and uses this as the denominator for other configurations. When $\alpha = 1$ this corresponds to a baseline Risk for all tradeoffs of the security model. This is because the β parameter and the cost of countermeasures are not included in the model.

Does Fashion produce good configurations?

Section 5.2.4 argues that combining the two security models would produce better configurations than having $\beta = 0$ or $\beta = 1$. This was confirmed in our experiments. Setting $\beta = 1$ produced a meaningful tradeoff between functionality and security. However, when we consider $0 < \beta < 1$ in the Hybrid manner, solution quality improves (e.g., $\alpha = .7$ improves functionality without impacting security). In all analyzed solutions, varying α with just the Path measure active ($\beta = 0$) produced solutions

that varied Path but not the actual Risk (other than blocking the gateway).

Thus, the setting of $\beta \in (0, 1)$ seems crucial, but the particular value within $(0, 1)$ does not seem to have a substantial effect on solution quality. This would be the case if one model is primarily being used as a tie-breaker for the other model. However, we cannot rule out that different settings of β would be preferable on different classes of networks and attack graphs. For the remainder of the analysis, consider $\beta = .5$, i.e., Path and Reach are equally weighted.

One can ask if FASHION produces solutions that trade off functionality and security. As a reminder, we use whitelist and source routing so setting $\alpha = 1$ corresponds to the minimum risk that is achievable while routing all desirable flows (assuming routing all flows is feasible within bandwidth constraints). Any further minimization of risk necessitates decreasing functionality. Furthermore, since we consider an external attacker, blocking all flows at the entry point is always the solution chosen when functionality is not considered (optimizing only over risk). Thus, every instance has functionality and risk of 0 when $\alpha = 0$. We remove this point from all analysis and consider $\alpha > 0$.

Since FASHION is approximating Risk it is possible for a decrease in α to lead to worse risk and functionality. However, in all of our experiments, risk and functionality were both monotonic for steps of α of size .1.[4] We consider 212 benchmarks of pod size of 6, each having 54 hosts. Varying α for each benchmark we plot both the functionality scores against α and also the risk scores against α. The functionality and risk scores are normalized as previously stated. Figure 5.4 demonstrates the probability mass functions over every instance solution varying $\alpha = .1$ to $\alpha = 1$ in

[4]We did observe small perturbations that violated monotonicity if one considered steps of α of .01.

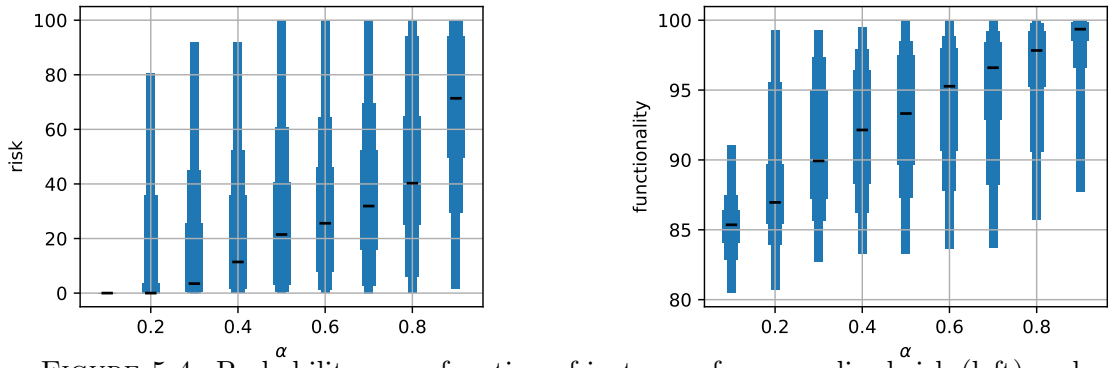


FIGURE 5.4: Probability mass function of instances for normalized risk (left) and normalized functionality (right). Start of bar represents min, widening at 10% and 25% with a line at 50%, reducing width at 75% and 90%, stopping at the max for each α. In all cases, risk is 0 when $\alpha = .1$.

steps of .1.

Note that the functionality scores across all benchmarks are relatively stable for $\alpha \geq .1$ while the risk scores vary significantly. Importantly, at every point on this curve FASHION is computing and outputting a corresponding network configuration. Note that in addition to considering what flows to include, the solution also describes how to route flow in a way that respects (and load balances) switch capacity. Based on visual inspection we classified our instances into three types of attack graphs.

Instances with the most area under the curve In such instances nodes with exploits to serve as an endpoint in many of the desired flows. In one generated pod 4 attack graph, a node with an exploit was the end point for a flow from 11 of the 15 other hosts. "Disconnecting" this node from the network required sacrificing many flows. This yields a sharp functionality vs. security tradeoff. Note such a graph can occur in practice when many clients need to access a critical, vulnerable resource such as a database.

Instances with the least area under the curve In such instances flows with high

value are mostly distinct from exploitable nodes. In one instance with 98 total flows, it is possible to achieve risk 0 by only blocking 15 flows. Such instances can be seen as easy: the risky nodes are not crucial to functionality.

Instances with many tradeoffs In such instances exploitable hosts have a meaningful but not overwhelming value of flow. In these cases, FASHION can mitigate risk in two ways: by severing external connections to prevent an attacker from entering the network, or by severing internal connections to prevent an attacker from moving laterally through network.

To illustrate this scenario and the corresponding choices, we consider one pod 4 attack graph with four exploitable nodes. These four nodes are all involved in both external and internal flows. Here the external flows were typically of higher value than that of internal flows, making FASHION sacrifice the internal flows for the sake of security at larger values of α. However as α decreases, external flows begin to be blocked which allows for previously severed internal flows to be serviced once again, as they are no longer needed to prevent lateral movement since the attacker cannot necessarily enter the network through external gateways. Table 5.2 shows the balance of external and internal flows blocked in this instance. In this instance, the larger sets of blocked internal flows at smaller values of α were not supersets of smaller sets of blocked internal flows seen at larger α values. This demonstrates an important capability of FASHION: the ability to recognize defenses whose current marginal cost (to functionality) exceeds their value (to security).

α	.9	.8	.7	.6	.5	.4	.3	.2	.1
External Flows Blocked	0	1	3	5	5	7	7	48	48
Internal Flows Blocked	3	11	11	15	15	23	23	0	0

TABLE 5.2: Number of external and internal flows blocked over varying values of α on example instance.

pod	6			8			10		12
#flows per host	3	5	10	3	5	10	1	3	1
min	13	25	65	152	466	1069	1650	1540	1256
25%	16	38	120	195	700	1545	2399	1608	1469
50%	19	43	151	250	763	1921	2686	1733	1853
75%	23	53	201	309	896	2259	2845	1990	2212
max	87	289	2316	607	2787	2825	2979	2308	2977
average	22	57	205	263	863	1937	2576	1836	1940

TABLE 5.3: Solve Time in Seconds for Pod 6, 8, 10 and 12 Networks with Different Flows Per Host with $\alpha = .7$

Does Fashion produce configurations in a timely manner?

To verify the scalability of FASHION and its ability to react to short term events, it is valuable to assess performance as a function of various input size parameters. Experiments were done on Fat-tree networks with pod sizes 6, 8, 10, and 12. Table 5.3 shows the model solve times when scaling the number of flows per host. In this experiment, the sizes of pods, number of flows as well as the number of exploits per host were increased. Rows labelled 25%, 50% and 75% report the percentile breakdown of the runtime for the instances considered. The columns vary the number of pods (6 to 12) as well as the number of flows per hosts.

The key observation here is that both the number of hosts and flows contribute significantly to the solve time. This is not surprising as the amount of hosts and flows affect the sizes of both the network and the attack graph, resulting in a large increase to the model size. We note that the average runtime does not always strictly increase

in Table 5.3. This is likely due to a smaller number of pod 10 & 12 instances being tested due to time constraints, contributing to higher variance among their reported times.

Another trend we observed is that the number of exploits alone can cause a rise in solve time, as it directly affects the size of the attack graph. Yet, the volume of exploits does not have as dramatic impact as the number of hosts/flows.

These results show that for networks of up to 128 hosts affected by a substantial number of flows and exploits, the framework produces optimal configurations within 3-7 minutes. Interestingly, Ingols et al. are able to scale to attack graphs with 40,000 hosts in a similar time period by introducing equivalence classes among hosts to reduce the size of the attack graph and achieve scalability [55]. Since the technique is orthogonal, it can easily be applied in our case as well. Homer et al. [51] build probabilistic attack graphs on networks with 100 hosts, their attack graph generation takes between 1-46 minutes depending on the complexity of the exploit chains. Both prior works generate attack graphs for *static network configuration* with no consideration of the functionality problem at all.

FASHION's response time (on networks of this scale) allows automated response to short term events such as publication of a zero-day before a patch is available. In an actual deployment, where the model must be solved repeatedly over time as inputs slightly evolve, the runtime can be drastically reduced when resolving the model by priming the optimization with the solution of the previous generation [100], [99].

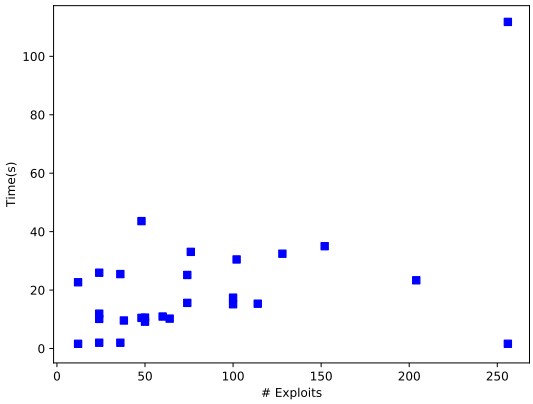


FIGURE 5.5: Need for linerizable risk measurements. Risk calculation time for output of FASHION on pod 8 instances using Python implementation of Wang et al.'s algorithm [121]. Pod 8 instances correspond to 128 hosts and 81 switches.

Can Risk be directly used?

Figure 5.5 shows the time required to compute the actual Risk, with $\alpha = .7$ and using our Python implementation of the algorithm by Wang et al. [121] of the network configuration output by FASHION. The instances considered use 128 hosts, with 12 to 250 vulnerabilities. Over this set of benchmarks, the computation time for the risk values in a fixed configuration can reach 100 seconds for 250 exploits. The time to evaluate the actual risk seems correlated to the number of exploits, and the evaluation algorithms start to struggle even with a relatively small number (e.g., 50 exploits). The search space associated with this kind of problem is generally huge. Using this algorithm specification of Risk would yield a highly non-linear formulation which would likely be intractable. Reverting to an enumeration of configurations and computing the risk a posteriori would be equally intractable. *Linearization is essential* to deliver a responsive and practical framework to assist network administrators.

5.5 Conclusion

Configuring Software Defined Networks to maximize the volume of customers data flows to and from servers while respecting device and link capacities is a classic flow optimization problem. Protecting such a network from adversaries attempting to exploit vulnerabilities that plague specific devices and hosts is equally important to address within organizations. Attack graphs are effective in modeling risk and finding mitigations (defensive measures). Unfortunately, risk and functionality are antagonistic objectives and optimizing one without caring for the other is unhelpful as it will deliver extreme solutions that are impractical. This paper considers both challenges in a holistic fashion and automatically computes new SDN configurations for network devices in response to emergent changes in demand, component risk or exploit discoveries. The FASHION framework models the customer demands, network devices and link capacities. It also captures two notions of risk, Path and Reach under an attack-dependency graph model within the overall optimization. The output from FASHION includes routing decisions for SDN devices as well as firewall mitigation decisions.

The paper demonstrates that FASHION can explore the trade-off between functionality and risk. As stated, FASHION optimizes over both objectives but the model can easily be converted into one where either security or functionality is a *constraint* and the other objective is optimized. The average of Path and Reach is an effective linearizable stand in for a risk calculation that is prohibitive to compute on the scale needed for a configuration search problem. Interestingly, the novel hybrid risk model enables FASHION to overcome their respective weaknesses and produce better solutions. From a practical angle, FASHION runs in matter of minutes on networks

of reasonable size (613 devices) and demonstrates potential for scalability. Finally, the empirical results indicate that the approximation adopted by FASHION does not jeopardize key properties such as monotonicity of functionality vs. risk.

The FASHION framework delivers a first step towards handling both functionality and risk for short-term response while producing consistent results with natural interpretations. Future directions include addressing not only source-routing but also destination routing within the network; handling scalability of the model size that currently depends on the number of network edges as well as the number of data flows; supporting a more varied set of controls beyond routing and blocking.

Bibliography

[1] "Iperf3-a tcp, udp, and sctp network bandwidth measurement tool."

[2] "Overview and principles of internet traffic engineering." [Online]. Available: https://tools.ietf.org/html/rfc3272

[3] S. Agarwal, M. Kodialam, and T. Lakshman, "Traffic engineering in software defined networks," in *2013 Proceedings IEEE INFOCOM*. IEEE, 2013, pp. 2211–2219.

[4] M. U. Aksu, K. Bicakci, M. H. Dilek, A. M. Ozbayoglu *et al.*, "Automated generation of attack graphs using NVD," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, vol. 2018-Janua. ACM, 2018, pp. 135–142.

[5] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1402958.1402967

[6] A. M. Al-Sadi, A. Al-Sherbaz, J. Xue, and S. Turner, "Routing algorithm optimization for software defined network wan," in *2016 Al-Sadeq International Conference on Multidisciplinary in IT and Communication Science and Applications (AIC-MITCSA)*. IEEE, 2016, pp. 1–6.

[7] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," *Computer Communication Review*, vol. 40, no. 4, pp. 63–74, 2010.

[8] H. M. Almohri, L. T. Watson, D. Yao, and X. Ou, "Security optimization of dynamic networks with probabilistic graph modeling and linear programming," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 4, pp. 474–487, 2016. [Online]. Available: http://www.ieee.org/publications{_}standards/publications/rights/index.html

[9] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2002.

[10] Z. Aslanyan and F. Nielson, "Pareto efficient solutions of attack-defence trees," in *International Conference on Principles of Security and Trust*. Springer, 2015, pp. 95–114.

[11] P. Baptiste, C. Le Pape, and W. Nuijten, *Constraint-Based Scheduling*. Kluwer Academic Publishers, 2001.

[12] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh, and P. H. Vance, "Branch-and-price: Column generation for solving huge integer programs," *Operations research*, vol. 46, no. 3, pp. 316–329, 1998.

[13] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "A general approach to network configuration verification," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 155–168.

[14] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker, "Don't mind the gap: Bridging network-wide objectives and device-level configurations," in *SIGCOMM 2016 - Proceedings of the 2016 ACM Conference on Special Interest Group on Data Communication*. Association for Computing Machinery, Inc, aug 2016, pp. 328–341.

[15] ——, "Network configuration synthesis with abstract topologies," in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2017, pp. 437–451.

[16] J. F. Benders, "Partitioning procedures for solving mixed-variables programming problems," *Numerische Mathematik*, vol. 4, no. 1, pp. 238–252, 1962.

[17] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 267–280. [Online]. Available: http://doi.acm.org/10.1145/1879141.1879175

[18] A. Binsahaq, T. R. Sheltami, and K. Salah, "A survey on autonomic provisioning and management of qos in sdn networks," *IEEE Access*, vol. 7, pp. 73 384–73 435, 2019.

[19] S. Bistarelli, F. Fioravanti, and P. Peretti, "Defense trees for economic evaluation of security investments," *Proc. - First Int. Conf. Availability, Reliab. Secur. ARES 2006*, vol. 2006, pp. 416–423, 2006.

[20] R. Bixby, M. Fenelon, Z. Gu, E. Rothberg, and R. Wunderling, *System Modelling and Optimization: Methods, Theory, and Applications*. Kluwer Academic Publishers, 2000, ch. MIP: Theory and practice – closing the gap, pp. 19–49.

[21] B. Brehmer, "The dynamic ooda loop: Amalgamating boyd's ooda loop and the cybernetic approach to command and control," in *Proceedings of the 10th international command and control research technology symposium*. Citeseer, 2005, pp. 365–368.

[22] G. Byeon, P. Van Hentenryck, R. Bent, and H. Nagarajan, "Communication-Constrained Expansion Planning for Resilient Distribution Systems," *ArXiv e-prints*, Jan. 2018.

[23] D. Callahan, T. Curry, D. Davidson, Z. T., B. Fuller, and L. Michel, "Functional and attack graph secured hybrid optimization of virtualized networks." [Online]. Available: https://github.com/devoncal77/FASHION

[24] D. Callahan, T. Curry, D. Davidson, H. Zitoun, B. Fuller, and L. Michel, "Fashion: Functional and attack graph secured hybrid optimization of virtualized networks," 2019. [Online]. Available: https://arxiv.org/abs/1910.07921

[25] D. Callahan, P. Shakarian, J. Nielsen, and A. N. Johnson, "Shaping operations to attack robust terror networks," in *2012 International Conference on Social Informatics*. IEEE, 2012, pp. 13–18.

[26] M. Capelle, S. Abdellatif, M.-J. Huguet, and P. Berthou, "Online virtual links resource allocation in software-defined networks," in *2015 IFIP Networking Conference (IFIP Networking)*. IEEE, 2015, pp. 1–9.

[27] Y. Cherdantseva, P. Burnap, A. Blyth, P. Eden, K. Jones, H. Soulsby, and K. Stoddart, "A review of cyber security risk assessment methods for scada systems," *Computers & security*, vol. 56, pp. 1–27, 2016.

[28] Cisco, "Cisco global cloud index 2016-21," 2018.

[29] T. Coatta and G. W. Neufeld, "Configuration management via constraint programming," in *CDS*. IEEE, 1992, pp. 90–101.

[30] G. Codato and M. Fischetti, "Combinatorial Benders' cuts for mixed-integer linear programming," *Operations Research*, vol. 54, no. 4, pp. 756–766, 2006. [Online]. Available: http://dx.doi.org/10.1287/opre.1060.0286

[31] A. Computing *et al.*, "An architectural blueprint for autonomic computing," *IBM White Paper*, vol. 31, no. 2006, pp. 1–6, 2006.

[32] M. Corp, "Common vulnerabilities and exposures," December 2018. [Online]. Available: https://cve.mitre.org

[33] T. Curry, D. Callahan, B. Fuller, and L. Michel, "DOCSDN: Dynamic and optimal configuration of software-defined networks," in *Australasian Conference on Information Security and Privacy*. Springer, 2019, pp. 456–474.

[34] G. B. Dantzig and P. Wolfe, "Decomposition principle for linear programs," *Oper. Res.*, vol. 8, no. 1, pp. 101–111, Feb. 1960. [Online]. Available: http://dx.doi.org/10.1287/opre.8.1.101

[35] C. Delimitrou, S. Sankar, A. Kansal, and C. Kozyrakis, "ECHO: Recreating network traffic maps for datacenters with tens of thousands of servers," *Pro-*

ceedings - 2012 IEEE International Symposium on Workload Characterization, IISWC 2012, pp. 14–24, 2012.

[36] R. Dewri, N. Poolsappasit, I. Ray, and D. Whitley, "Optimal security hardening using multi-objective optimization on attack tree models of networks," in *Proceedings of the 14th ACM conference on Computer and communications security - CCS '07*. New York, New York, USA: ACM Press, 2007, p. 204. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1315245.1315272

[37] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, "Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks," in *Proceedings of the 2012 Asia Pacific signal and information processing association annual summit and conference*. IEEE, 2012, pp. 1–8.

[38] A. El-Hassany, P. Tsankov, L. Vanbever, M. Vechev, and E. Zürich, "NetComplete: Practical Network-Wide Configuration Synthesis with Autocompletion," Tech. Rep.

[39] S. K. Fayaz, Y. Tobioka, V. Sekar, M. Bailey, and M. Bailey, "Bohatei : Flexible and Elastic DDoS Defense This paper is included in the Proceedings of the," 2015.

[40] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: Flexible and elastic DDoS defense." in *USENIX Security Symposium*, 2015, pp. 817–832.

[41] FIRST, "Common vulnerability scoring system," June 2019. [Online]. Available: https://www.first.org/cvss/

[42] C. Fishman, "The system that actually worked," May 2020. [Online]. Available: https://www.theatlantic.com/ideas/archive/2020/05/miracle-internet-not-breaking/611212/

[43] N. Foster, M. J. Freedman, R. Harrison, J. Rexford, M. L. Meola, and D. Walker, "Frenetic: a high-level language for openflow networks," in *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*. ACM, 2010, p. 6.

[44] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," *ACM Sigplan Notices*, vol. 46, no. 9, pp. 279–291, 2011.

[45] B. Fourer, "Amazing solver speedups," online, 2015, http://bob4er.blogspot.com/2015/05/amazing-solver-speedups.html.

[46] M. Frigault and L. Wang, "Measuring network security using bayesian network-based attack graphs," *Proceedings - International Computer Software and Applications Conference*, pp. 698–703, 2008.

[47] A. Gangwal, M. Gupta, M. S. Gaur, V. Laxmi, and M. Conti, "Elba: Efficient layer based routing algorithm in sdn," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2016, pp. 1–7.

[48] P. Gill, M. Schapira, and S. Goldberg, "A survey of interdomain routing policies," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 1, pp. 28–34, 2013.

[49] H. Hijazi, T. W. K. Mak, and P. Van Hentenryck, "Power system restoration with transient stability," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, ser. AAAI'15. AAAI Press, 2015, pp. 658–664. [Online]. Available: http://dl.acm.org/citation.cfm?id=2887007.2887099

[50] J. Homer, X. Ou, and D. Schmidt, "A sound and practical approach to quantifying security risk in enterprise networks," *Kansas State University Technical Report*, pp. 1–15, 2009.

[51] J. Homer, S. Zhang, X. Ou, D. Schmidt, Y. Du, S. R. Rajagopalan, and A. Singhal, "Aggregating vulnerability metrics in enterprise networks using attack graphs," *Journal of Computer Security*, vol. 21, no. 4, pp. 561–597, 2013.

[52] J. Hooker, "Logic-based Benders decomposition," *Mathematical Programming*, vol. 96, p. 2003, 1995.

[53] ——, *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction.* John Wiley and Sons, 2000.

[54] H. Huang, S. Zhang, X. Ou, A. Prakash, and K. Sakallah, "Distilling critical attack graph surface iteratively through minimum-cost sat solving," in *Proceedings of the 27th Annual Computer Security Applications Conference.* ACM, 2011, pp. 31–40.

[55] K. Ingols, M. Chu, R. Lippmann, S. Webster, and S. Boyer, "Modeling modern network attacks and countermeasures using attack graphs," in *2009 Annual Computer Security Applications Conference.* IEEE, 2009, pp. 117–126.

[56] K. Ingols, R. Lippmann, and K. Piwowarski, "Practical attack graph generation for network defense," in *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*. IEEE, 2006, pp. 121–130.

[57] J. Ioannidis and S. M. Bellovin, "Pushback: Router-based defense against DDoS attacks," 2001.

[58] ——, "Implementing pushback: Router-based defense against DDoS attacks." in *NDSS*, vol. 2, 2002.

[59] W. Jansen, *Directions in security metrics research*. Diane Publishing, 2010.

[60] S. Jha, O. Sheyner, and J. Wing, "Two formal analyses of attack graphs," in *Computer Security Foundations Workshop, 2002. Proceedings. 15th IEEE*. IEEE, 2002, pp. 49–63.

[61] S. Kandula, "Inside the Social Network ' s (Datacenter) Network – Public Review," pp. 123–137.

[62] K. Kaynar, "A taxonomy for attack graph generation and usage in network security," *Journal of Information Security and Applications*, vol. 29, pp. 27–56, 2016.

[63] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[64] M. H. Khouzani, Z. Liu, and P. Malacaria, "Scalable min-max multi-objective cyber-security optimisation over probabilistic attack graphs," *European Journal of Operational Research*, vol. 278, no. 3, pp. 894–903, 2019. [Online]. Available: https://doi.org/10.1016/j.ejor.2019.04.035

[65] A. Khurshid, W. Zhou, M. Caesar, and P. Godfrey, "Veriflow: Verifying network-wide invariants in real time," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 49–54.

[66] H. Kim, J. Reich, A. Gupta, M. Shahbaz, N. Feamster, and R. J. Clark, "Kinetic: Verifiable dynamic network control." in *NSDI*, 2015, pp. 59–72.

[67] B. Kordy, L. Piètre-Cambacédès, and P. Schweitzer, "DAG-based attack and defense modeling: Don't miss the forest for the attack trees," 2014.

[68] S. Kottler. (2018, March) February 28th ddos incident report. [Online]. Available: https://githubengineering.com/ddos-incident-report/

[69] P. Kotzias, L. Bilge, P.-A. Vervier, and J. Caballero, "Mind your own business: A longitudinal study of threats and vulnerabilities in enterprises." in *NDSS*, 2019.

[70] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[71] A. Kucminski, A. Al-Jawad, P. Shah, and R. Trestian, "Qos-based routing over software defined networks," in *2017 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*. IEEE, 2017, pp. 1–6.

[72] E. Lam and P. V. Hentenryck, "A branch-and-price-and-check model for the vehicle routing problem with location congestion," *Constraints*,

vol. 21, no. 3, pp. 394–412, Jul. 2016. [Online]. Available: http://dx.doi.org/10.1007/s10601-016-9241-2

[73] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 1–6.

[74] S. Layeghy, F. Pakzad, and M. Portmann, "Scor: Software-defined constrained optimal routing platform for sdn," *arXiv preprint arXiv:1607.03243*, 2016.

[75] ——, "SCOR: software-defined constrained optimal routing platform for SDN," *CoRR*, vol. abs/1607.03243, 2016. [Online]. Available: http://arxiv.org/abs/1607.03243

[76] A. Lenin, O. Gadyatskaya, D. Ionita, W. Pieters, A. Tanner, S. Saraiva, C. Muller, J. Willemson, M. Ford, and S. Muller, "Technology-supported risk estimation by predictive assessment of socio-technical security," 2016.

[77] C. Lin, K. Wang, and G. Deng, "A qos-aware routing in sdn hybrid networks," *Procedia Computer Science*, vol. 110, pp. 242–249, 2017.

[78] R. Lippmann and K. Ingols, "An annotated Review of Past Papers on Attack Graphs," 2005.

[79] R. P. Lippmann and J. F. Riordan, "Threat-based risk assessment for enterprise networks," *Lincoln Laboratory Journal*, vol. 22, no. 1, pp. 33–45, 2016.

[80] R. Lippmann, J. Riordan, T. Yu, and K. Watson, "Continuous security metrics for prevalent network threats: introduction and first four metrics," Massachusetts Inst of Tech Lexington Lincoln Lab, Tech. Rep., 2012.

[81] R. Majumdar and V. Kunčak, Eds., *Computer Aided Verification*, ser. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, vol. 10427. [Online]. Available: http://link.springer.com/10.1007/978-3-319-63390-9

[82] B. Marczak, N. Weaver, J. Dalek, R. Ensafi, D. Fifield, S. McKune, A. Rey, J. Scott-Railton, R. Deibert, and V. Paxson, "China's great cannon," *Citizen Lab*, vol. 10, 2015.

[83] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[84] J. Meza, T. Xu, K. Veeraraghavan, and O. Mutlu, "A large scale study of data center network reliability," in *Proceedings of the Internet Measurement Conference 2018*, 2018, pp. 393–407.

[85] J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.

[86] MITRE, "Common weakness scoring system," December 2018. [Online]. Available: https://cwe.mitre.org/cwss/cwss_v1.0.1.html/

[87] J. T. Moy, *OSPF: anatomy of an Internet routing protocol*. Addison-Wesley Professional, 1998.

[88] H. Nagarajan, E. Yamangil, R. Bent, P. V. Hentenryck, and S. Backhaus, "Optimal resilient transmission grid design," in *PSCC*. IEEE, 2016, pp. 1–7.

[89] S. Narain, G. Levin, S. Malik, and V. Kaul, "Declarative infrastructure configuration synthesis and debugging," in *Journal of Network and Systems Management*, vol. 16, no. 3, sep 2008, pp. 235–258.

[90] P. Neves, R. Calé, M. R. Costa, C. Parada, B. Parreira, J. Alcaraz-Calero, Q. Wang, J. Nightingale, E. Chirivella-Perez, W. Jiang *et al.*, "The SELFNET approach for autonomic management in an NFV/SDN networking paradigm," *International Journal of Distributed Sensor Networks*, vol. 12, no. 2, p. 2897479, 2016.

[91] NIST, "Cybersecurity framework," April 2018. [Online]. Available: https://www.nist.gov/cyberframework/framework

[92] ——, "National vulnerability database," December 2018. [Online]. Available: https://nvd.nist.gov

[93] G. Optimization, "Inc.,"gurobi optimizer reference manual," 2015," *URL: http://www. gurobi. com*, 2014.

[94] X. Ou, W. F. Boyer, and M. A. McQueen, "A scalable approach to attack graph generation," in *Proceedings of the 13th ACM conference on Computer and communicati ons security*. ACM, 2006, pp. 336–345.

[95] T. Peng, C. Leckie, and K. Ramamohanarao, "Survey of network-based defense mechanisms countering the DoS and DDoS problems," *ACM Computing Surveys (CSUR)*, vol. 39, no. 1, p. 3, 2007.

[96] W. Pieters, D. Hadziosmanovic, A. Lenin, L. Montoya, and J. Willemson, "Trespass: plug-and-play attacker profiles for security risk analysis," *IEEE Security & Privacy poster abstracts*, 2014.

[97] K. Piwowarski, "Evaluating and Strengthening Enterprise," *Network*, no. October, 2005.

[98] N. Poolsappasit, R. Dewri, and I. Ray, "Dynamic security risk management using bayesian attack graphs," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 1, pp. 61–74, 2011.

[99] S. M. Pour, J. H. Drake, L. S. Ejlertsen, K. M. Rasmussen, and E. K. Burke, "A hybrid constraint programming/mixed integer programming framework for the preventive signaling maintenance crew scheduling problem," *European Journal of Operational Research*, vol. 269, no. 1, pp. 341 – 352, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221717307646

[100] T. Ralphs and M. Güzelsoy, "Duality and warm starting in integer programming," in *The Proceedings of the 2006 NSF Design, Service, and Manufacturing Grantees and Research Conference*, 2006.

[101] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, "Modular SDN programming with Pyretic," *Technical Reprot of USENIX*, 2013.

[102] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 323–334, 2012.

[103] R. W. Ritchey and P. Ammann, "Using model checking to analyze network vulnerabilities," *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 156–165, 2000.

[104] F. Rossi, P. v. Beek, and T. Walsh, *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. New York, NY, USA: Elsevier Science Inc., 2006.

[105] R. E. Sawilla and X. Ou, "Identifying critical attack assets in dependency attack graphs," in *European Symposium on Research in Computer Security*. Springer, 2008, pp. 18–34.

[106] B. Schneier, "Attack trees," *Dr. Dobb's journal*, vol. 24, no. 12, pp. 21–29, 1999.

[107] G. Schryen, "Is open source security a myth? what do vulnerability and patch data say?" *Communications of the ACM (CACM)*, vol. 54, no. 5, pp. 130–139, 2011.

[108] D. A. Schult, "Exploring network structure, dynamics, and function using networkx," in *In Proceedings of the 7th Python in Science Conference (SciPy*, 2008, pp. 11–15.

[109] P. Shakarian, P. Roos, D. Callahan, and C. Kirk, "Mining for geographically disperse communities in social networks by leveraging distance modularity," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '13. New York, NY, USA: ACM, 2013, pp. 1402–1409. [Online]. Available: http://doi.acm.org/10.1145/2487575.2488194

[110] P. Shakarian, G. I. Simari, and D. Callahan, "Reasoning about complex networks: a logic programming approach," MILITARY ACADEMY WEST POINT NY, Tech. Rep., 2013.

[111] P. Shaw, "Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems," in *Proceedings of Fourth International Conference on the Principles and Practice of Constraint Programming (CP'98)*. Springer Verlag, October 1998, pp. 417–431.

[112] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *Security and privacy, 2002. Proceedings. 2002 IEEE Symposium on*. IEEE, 2002, pp. 273–284.

[113] A. Singhal and X. Ou, "Security risk analysis of enterprise networks using probabilistic attack graphs," in *Network Security Metrics*. Springer, 2017, pp. 53–73.

[114] R. Skowyra, A. Lapets, A. Bestavros, and A. Kfoury, "A verification platform for SDN-enabled applications," in *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. IEEE, 2014, pp. 337–342.

[115] S. Stolfo, S. M. Bellovin, and D. Evans, "Measuring security," *IEEE Security & Privacy*, vol. 9, no. 3, pp. 60–65, 2011.

[116] G. Stoneburner, A. Y. Goguen, and A. Feringa, "SP 800-30. risk management guide for information technology systems," 2002.

[117] K. Subramanian, L. D'Antoni, and A. Akella, "Genesis: Synthesizing forwarding tables in multi-tenant networks," in *Conference Record of the Annual ACM*

Symposium on Principles of Programming Languages. Association for Computing Machinery, jan 2017, pp. 572–585.

[118] S. Tomovic, N. Prasad, and I. Radusinovic, "Sdn control framework for qos provisioning," in *2014 22nd Telecommunications Forum Telfor (TELFOR)*. IEEE, 2014, pp. 111–114.

[119] S. Tomovic and I. Radusinovic, "Fast and efficient bandwidth-delay constrained routing algorithm for sdn networks," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. IEEE, 2016, pp. 303–311.

[120] N. L. Van Adrichem, C. Doerr, and F. A. Kuipers, "Opennetmon: Network monitoring in openflow software-defined networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–8.

[121] L. Wang, S. Jajodia, and A. Singhal, *Network security metrics*, 2017.

[122] R. Wang, D. Butnariu, J. Rexford *et al.*, "Openflow-based server load balancing gone wild." *Hot-ICE*, vol. 11, pp. 12–12, 2011.

[123] Z. Wang and J. Crowcroft, "Quality-of-service routing for supporting multimedia applications," *IEEE Journal on selected areas in communications*, vol. 14, no. 7, pp. 1228–1234, 1996.

[124] C. Xu, B. Chen, P. Fu, and H. Qian, "A dynamic resource allocation model for guaranteeing quality of service in software defined networking based cloud computing environment," in *International Conference on Cloud Computing and Security*. Springer, 2015, pp. 206–217.

[125] C. Yu, J. Lan, Z. Guo, and Y. Hu, "Drom: Optimizing the routing in software-defined networks with deep reinforcement learning," *IEEE Access*, vol. 6, pp. 64 533–64 539, 2018.

[126] R. Yu, G. Xue, V. T. Kilari, and X. Zhang, "Deploying robust security in internet of things," in *IEEE Conference on Computer and Network Security*, 2018.

[127] T.-F. Yu, K. Wang, and Y.-H. Hsu, "Adaptive routing for video streaming with qos support over sdn networks," in *2015 International Conference on Information Networking (ICOIN)*. IEEE, 2015, pp. 318–323.

[128] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks," *IEEE communications surveys & tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.

[129] S. Zhang and S. Malik, "SAT based verification of network data planes," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2013, pp. 496–505.

[130] S. Zhang, X. Zhang, and X. Ou, "After we knew it," pp. 317–328, 2014.

[131] X. Zhang, W. Hou, L. Guo, Q. Zhang, P. Guo, and R. Li, "Joint optimization of latency monitoring and traffic scheduling in software defined heterogeneous networks," *Mobile Networks and Applications*, vol. 25, no. 1, pp. 102–113, 2020.

[132] Q. Zhou, K. Wang, P. Li, D. Zeng, S. Guo, B. Ye, and M. Guo, "Fast Coflow Scheduling via Traffic Compression and Stage Pipelining in Datacenter Networks," *IEEE Transactions on Computers*, vol. PP, no. c, pp. 1–1, 2019.